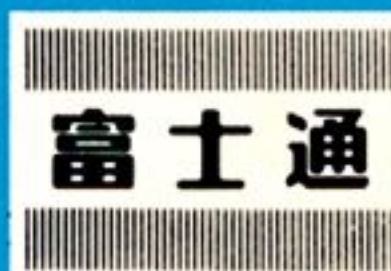
82SM-000050-1

パーソナルコンピュータ

FM-7

アブソリュートアセンブラ解説書

富士通

はじめに

このたびはFM−7用アブソリュートアセンブラをお買い上げいただきありがとうございます.

本書はパーソナルコンピュータFM−7のF−BASIC Ver 3.0の管理下で動作するMBL6809用アセンブラ(アブソリュートアセンブラ)について解説したものです.

アセンブラ言語は機械語に1対1で対応した記号言語であり，BASICでは無理の多かったきめのこまかい処理も効率よく記述することができます．また，アセンブラプログラムは直接機械語に変換されてから，実行させるためにBASICなどとくらべてより高速です．BASICプログラムとアセンブラプログラムとの長所をうまく組合せることによって，よりコンパクトで高速なプログラムの作成が可能となるでしょう.

昭和58年6月

目　　次

第1章　概　　要 ………………………… 1

1.1　特　　長 ………………………… 5

1.2　本書の記法 ………………………… 6

第2章　文の形式 ………………………… 9

2.1　行番号欄 ………………………… 9

2.2　引用符欄 ………………………… 10

2.3　ラベル欄 ………………………… 10

2.4　命　令　欄 ………………………… 10

2.5　オペランド欄 ………………………… 11

2.6　注　釈　欄 ………………………… 12

第3章　言語の基本規則 ………………………… 13

3.1　文字セット ………………………… 13

3.2　番　　地 ………………………… 14

3.3　項 ………………………… 15

3.3.1　記　　号 ………………………… 15

3.3.2　ロケーションカウンタ参照 ………………………… 16

3.3.3　自己規定項 ………………………… 17

3.4　式 ………………………… 20

3.4.1　式に関する規則 ………………………… 21

3.4.2　2文字演算子 ………………………… 21

3.4.3　式の値の決定 ………………………… 22

3.5 簡略命令コード …… 22

第4章 機械命令 …… 25

4.1 機械命令の形式 …… 25

4.2 番地指定 …… 26

4.2.1 インヘレント番地指定 …… 26

4.2.2 アキュムレータ番地指定 …… 27

4.2.3 イミーディエイト番地指定 …… 28

4.2.4 インデックス番地指定 …… 29

4.2.5 相対番地指定 …… 40

4.2.6 直接番地指定と拡張番地指定 …… 43

4.2.7 レジスタ番地指定 …… 46

4.3 命令形式 …… 49

第5章 制御命令 …… 51

5.1 NAM 命令 …… 51

5.2 END 命令 …… 51

5.3 ORG 命令 …… 52

5.4 SETDP 命令 …… 53

5.5 OPT 命令 …… 54

5.6 TTL 命令 …… 56

5.7 PAGE (PAG) 命令 …… 57

5.8 SPC 命令 …… 57

第6章　記号定義命令 …… 59

6.1　EQU 命令 …… 59

6.2　REG 命令 …… 60

第7章　データと領域の定義 …… 63

7.1　FCC 命令 …… 63

7.2　FCB 命令 …… 64

7.3　FDB 命令 …… 65

7.4　BSZ 命令 …… 66

7.5　RMB 命令 …… 67

第8章　使用手引 …… 69

8.1　機器構成 …… 69

8.2　翻訳の手順 …… 70

8.2.1　翻訳の操作手順 …… 70

8.2.2　操作手順中に出力されるエラーメッセージ …… 74

8.3　ソースプログラムファイルの作成 …… 75

8.4　アセンブラプログラムのコピー方法 …… 78

8.5　アセンブルリストの種類 …… 78

8.5.1　ソースプログラム及び目的プログラムリスト …… 78

8.5.2　記号テーブルリスト …… 80

8.6　エラー及び警告メッセージ …… 80

8.6.1　エラーメッセージ …… 80

8.6.2　警告メッセージ …… 85

8.7　翻訳時のメモリ配置 …… 86

8.8　翻訳処理能力 …… 87

付　録

付録1　機械命令一覧表 …………………………………………… 89
付録2　ポストバイトの形式 ………………………………………103
付録3　アセンブラ命令一覧表 ……………………………………105
付録4　裏RAM機能 ……………………………………………………107
付録4.1　裏RAMについて ……………………………………107
付録4.2　〃　制御回路 ………………………………………108
付録4.3　〃　にデータやプログラムを転送する方法 ………109
付録4.4　〃　のサブルーチンを使用する方法 ………………113
付録4.5　BIOSを使用する場合 …………………………………115

第1章　概　要

アセンブラとは，CPUが直接実行することのできる機械語のプログラムを作成するためのプログラム言語です．パーソナルコンピュータで一般に用いられているBASICの大半はBASICインタプリタと呼ばれる形式のプログラム言語です．インタプリタではプログラマによって書かれたプログラムがそのままの形か，少なくとも元のプログラムに復元できる範囲内で圧縮された形（中間コード）にて記憶され，1命令（ステートメント）づつ取り出されて実行されます．BASICはそれ自体完成されたすばらしい言語の1つですがBASICインタプリタという処理プログラム（BASICのシステムプログラム）を通して実行されるために，プログラムの記述上におけるいくつかの制約や速度上の問題点（BASICインタプリタは他のコンパイラ言語やアセンブラ言語にくらべて一般的に遅い）があることも事実です．

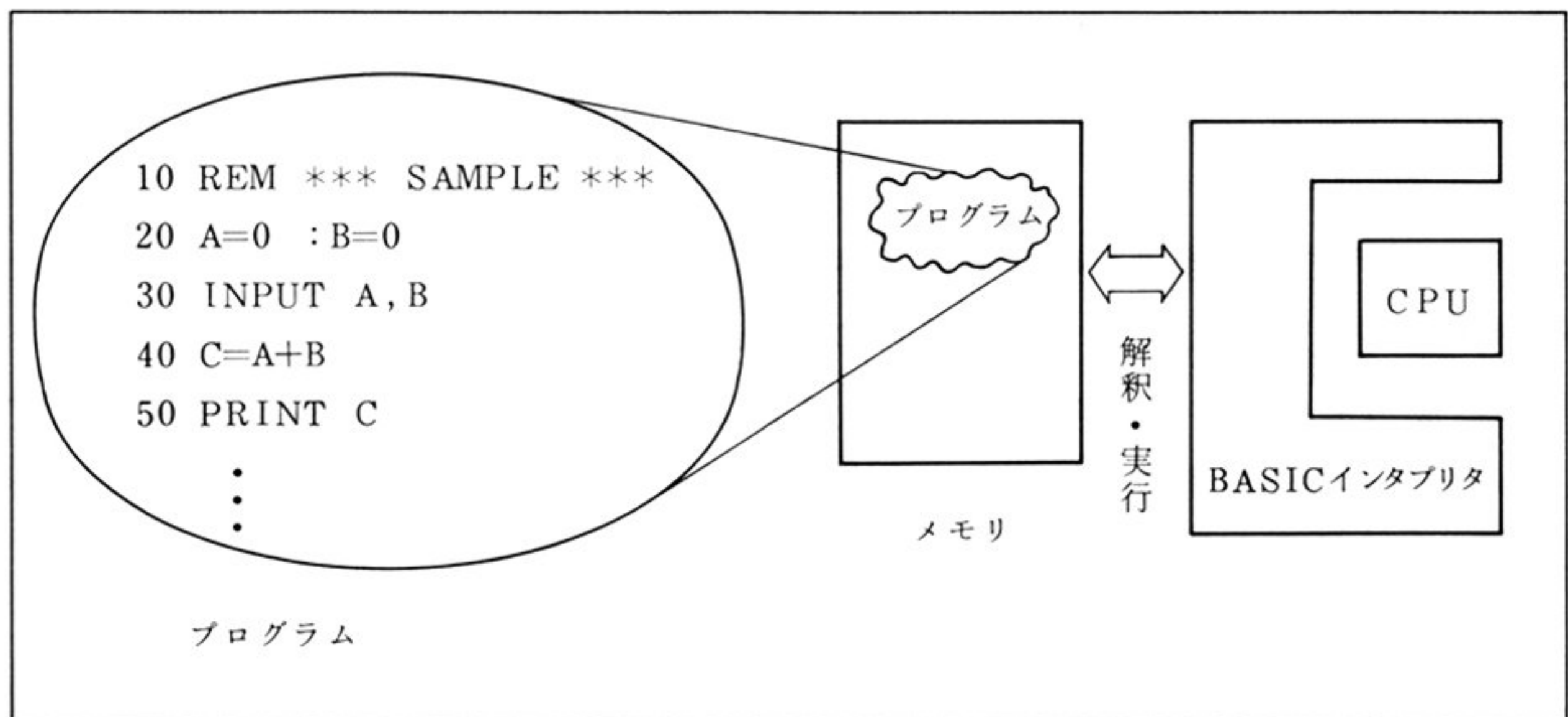



図1.1　BASICインタプリタのイメージ

機械語によるプログラムはパーソナルコンピュータのハードウェア（CPU）によって直接実行されるためにその処理速度はたいへん速くなります．有能なプログラマがある目的をもって作成したプログラムはおそらく他のどんな言語（アセンブラを除く）によって作成されたプログラムより高速で動作するでしょう．けれども機

械語のプログラムは，プログラムがコンピュータのメモリの中に入っているのと同じ状態つまりビットのON（1）とOFF（0）の組み合わせ（数字の羅列）にて表現されるために，人間にとってたいへん分りづらく，プログラムの作成はとても困難です．

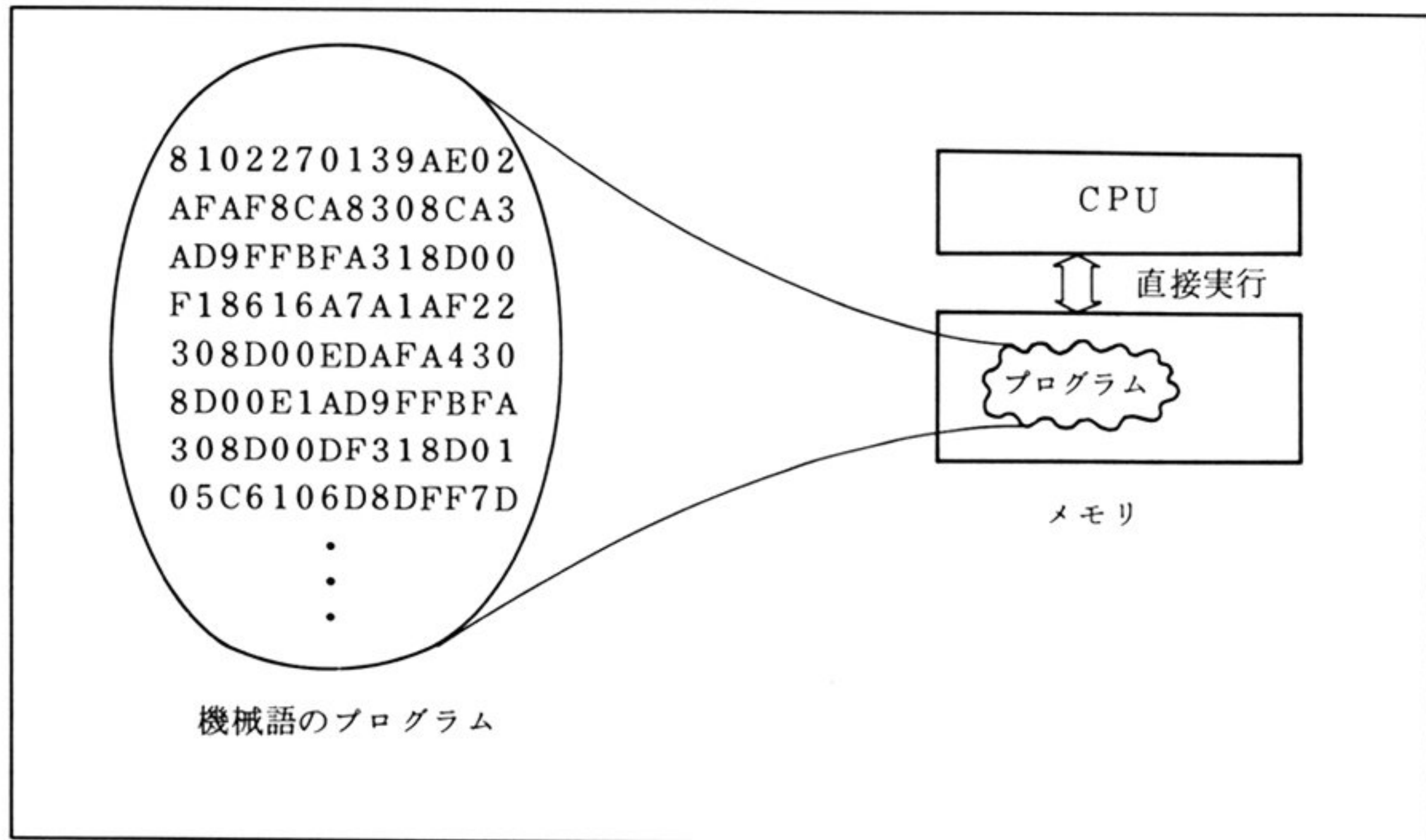



図1.2　機械語プログラム

アセンブラ言語は機械語のプログラムの作成をわかりやすく容易なものとするために機械語を記号化して覚えやすくかつ考えやすくしたものです．アセンブラはCPUの命令をニーモニックコードと呼ばれる3～5文字の文字列にて表現します．またメモリ上の位置も数値で表わされるメモリアドレスではなく英数文字列による記号番地にて表現します．（もちろんメモリ番地でも表現できます．）

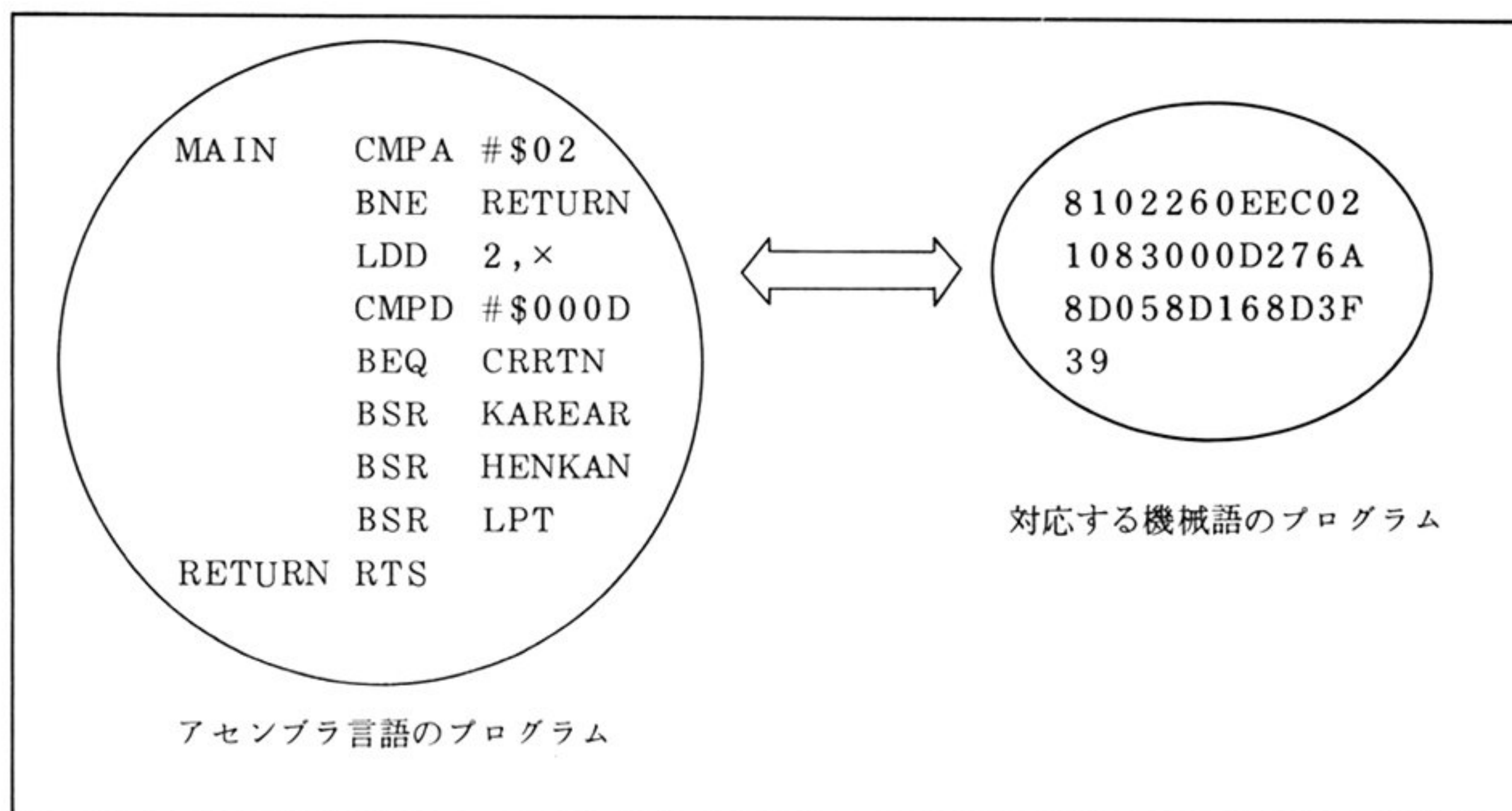



図1.3 アセンブラ言語と機械語の対応

FM−7のアブソリュートアセンブラ（本アセンブラ）はFM−7のF−BASIC Ver 3.0のもとで動作するアセンブラです．プログラマ（ユーザ）はF−BASICのスクリーンエディタを用いてBASICのプログラムと同様にアセンブラプログラムを作成することができます．アセンブラ自身もF−BASICにて呼び出され，ユーザはRUN "ASM09" とタイプするだけでアブソリュートアセンブラを起動することができます．ユーザの作成したアセンブラプログラム（ソースプログラム）は機械語に変換された後に，自動的にF−BASICの機械語ファイルとしてセーブされます．したがって作成した機械語プログラムはF−BASICのLOADMコマンドにて直接ロードしてEXEC命令やユーザ関数命令（USRn）にて実行することができます．

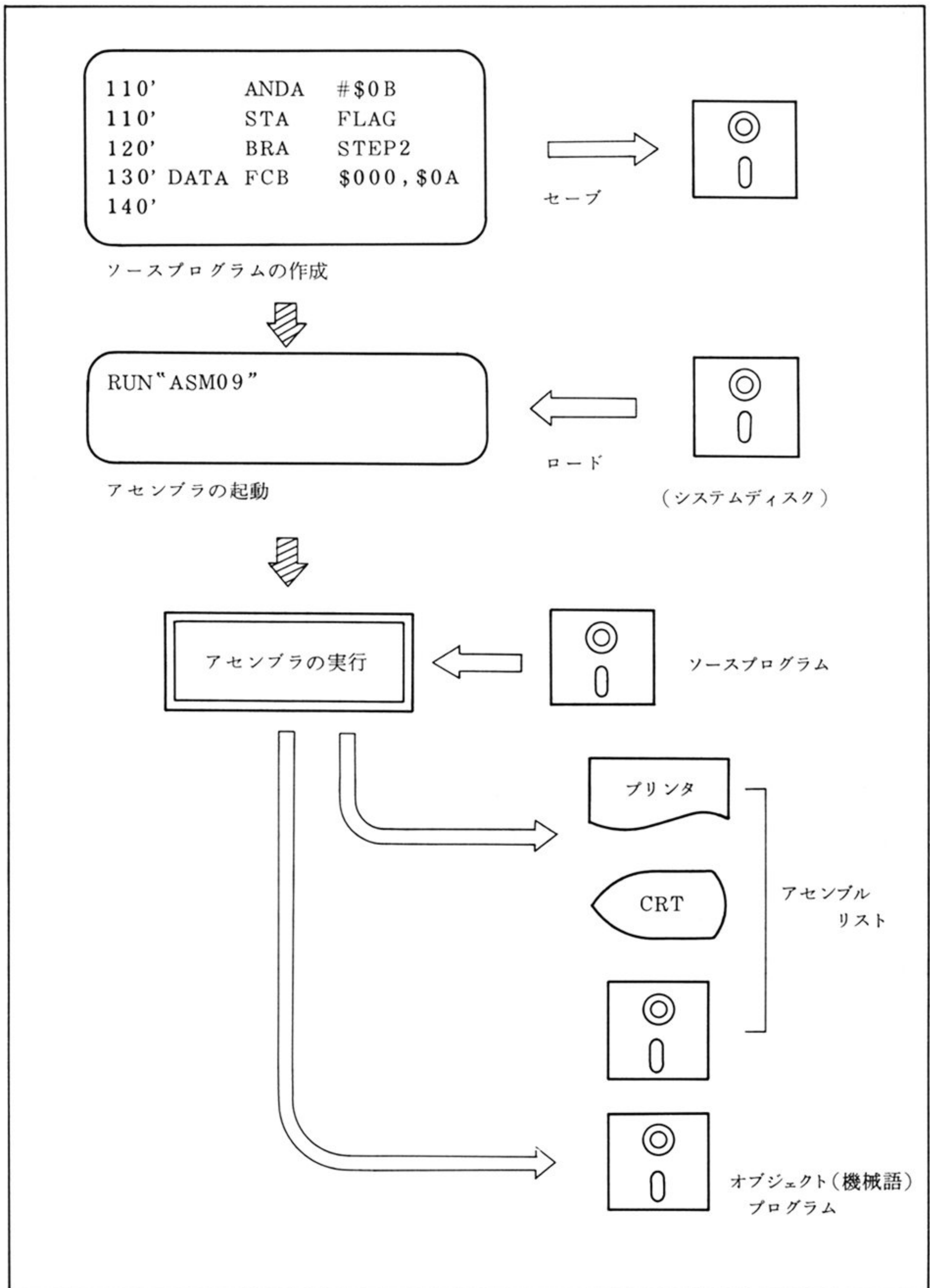



図1.4　プログラム作成の流れ

1.1 特　　長

アセンブラ言語でプログラミングする場合の利点は，機械語にもっとも近い記号言語なので，高級言語に比較して，ビットやバイト単位のきめ細い処理を容易に行なうことができることです.

アセンブラ言語には，次にあげる2種類の命令があります.

・機械命令（実行命令とも呼ぶ）

・アセンブラ命令

アセンブラ言語では，上記2種類の命令を表現する単位を文と呼び，記述上の見やすさを考慮した注釈文を加えて図1.5のような文の体系をとっています.

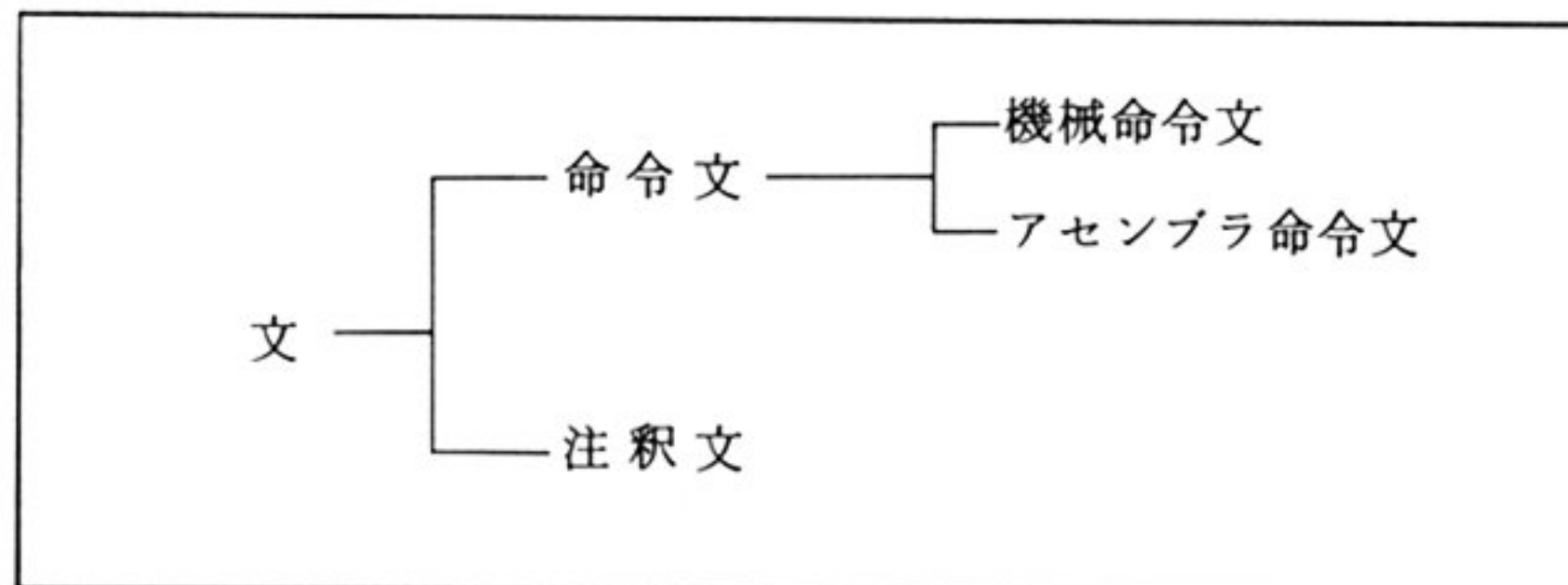



図1.5　アセンブラ言語の文の体系

(1)　機械命令文

マイクロプロセッサMBL6809の機械語に1対1に対応する命令文の一つで機械語の働きを記号化したものであり，これを機械命令文と呼びます.

機械命令文は，アセンブラ言語の基本的な文であり，「第4章　機械命令」で説明します.

(2)　アセンブラ命令文

アセンブラが，ソースプログラムから目的プログラムに翻訳する途中で，アセンブラがある種の働きをするように指示する文です.

アセンブラ命令文は，機械命令を使用してプログラムを作成するときの補助的な役割を果たすもので

・翻訳の制御

・番地指定
・記号の定義
・データと領域の定義
・リストの制御

を行なうものであり，これらの命令の中で，目的プログラムの一部として出力されるものであります．アセンブラ命令は，「第5章　制御命令」から「第7章　データと領域の定義」で説明します．

(3) 注 釈 文

注釈文は，文の先頭がアスタリスク(*)から始まっている文であり，プログラム中の任意の位置（行）に書くことができます．注釈文はプログラマの便宜のためだけに使用され，アセンブラの動作および作成されるプログラムには何も影響しません．

1.2 本 書 の 記 法

「第4章　機械命令」以降の各章では，文の説明を次に述べる記述上の規則に従って行ないます．

(1) 文の表現方法

文（図1.5を参照）の記述形式を図1.6に示します．

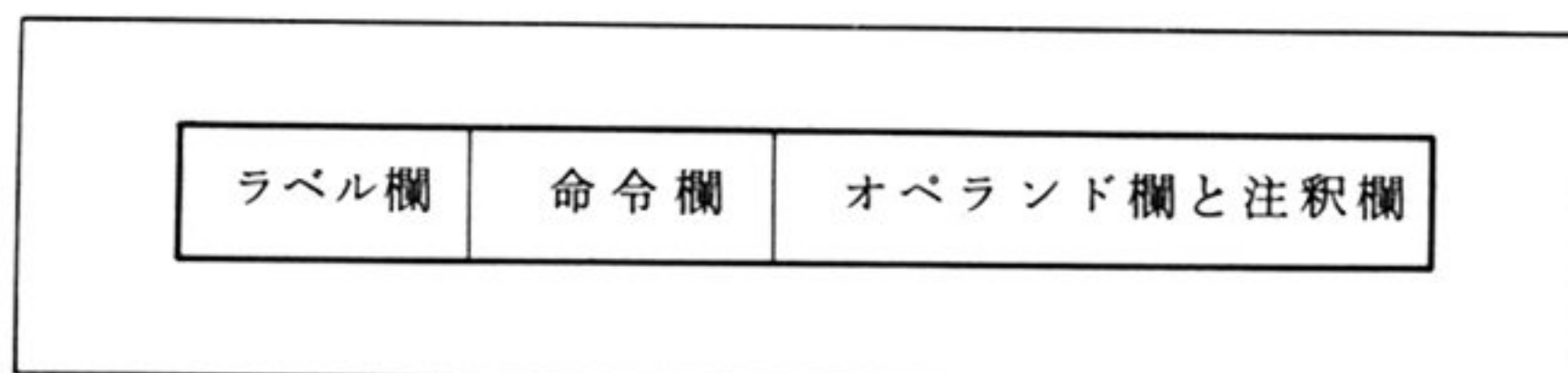



図1.6　文 の 表 現

(2) 各欄の記述記号

各欄の説明を行なうためにa～dの記号を使用します．

a. ［　］： 任意選択を示す記号です．

例えば［A］は，何も書かないか，あるいはAと書くことを意味

します．

b．｛　｝：択一選択を示します．

例えば｛$\genfrac{}{}{0pt}{}{A}{B}$｝と書けば，ＡかＢのうち一つを選んで書くことを意味します．

c．………：（点線）同形式の項目を複数回記述することが可能な場合，点線の直前におかれている項目の反復を示します．

d．　　：各欄に何も書かれない場合は，その欄に指定する項目がないことを示します．

第2章　文の形式

本章では，アセンブラ言語を使用してプログラムを作成する場合の文の形式について説明します．

アセンブラ言語では，文は一行に記述され，一行は次の6つの欄から構成されます．

(1) 行番号欄
(2) 引用符欄
(3) ラベル欄（ラベルフィールド）
(4) 命令欄（オペレーションフィールド）
(5) オペランド欄（オペランドフィールド）
(6) 注釈欄（コメントフィールド）

図2.1に記述例を示します．

```
100'* COMMENT LINE                                      ①
110'        LDX    O,X      IX-REG SET                  ②
120'        LDA    FLAG     LOAD FLAG BYTE              ③
130'        AND    #$03     CHACK FLAG BIT
140'        BNE    LAB2     BRANCH IF NOT EQUAL
150'        BSR    SUB1     CALL SUBROUTINE
160'LAB2    STA    DATA                                 ④
```


図2.1　文の記述例

2.1 行 番 号 欄

行番号は，プログラマが各行を他の行と区別するために使用します．行番号は，一つの行の1けた目から始まり，5けた以内の10進数で構成されます（ただし，

その値は65529より小さくならなければならない）．各行番号は昇順に指定します．

2.2 引用符欄

引用符欄には，文の始まりを示す引用符(')を必ず記述してください．引用符欄は，行番号欄から1けた以上あけた所から始まります．

2.3 ラベル欄

ラベル欄は，引用符欄の次から始まり，次にあげる三つの形式があります．

a．最初の文字がアスタリスク(*)のときは，この行が注釈行であることを示します．アセンブラにとって，リストすることを除いて意味を持ちません．
（例．図2.1の①の行）

b．最初の文字が空白のときは，ラベルを持たない行であることを示します．
（例．図2.1の②もしくは③の行）

c．記号（例．図2.1の④の行）

2.4 命令欄

命令欄は，一つの行のラベル欄の直後から始まります．この欄は，1～6文字の命令コードより構成されます．ラベル欄に記号を記述した場合には，命令欄との間に1個以上の空白を指定します．

命令欄には，次にあげる二つの形式があります．

a．簡略命令コード（ニーモニックオペレーションコード）

これらは，マイクロプロセッサMBL6809の機械命令に相当します．簡略命令コードについては，「付録1　機械命令一覧表」を参照してください．

この命令欄では，アキュムレータ番地指定形式の場合は，命令コードに続いてAもしくはB（アキュムレータを示す）を書くことができます．

（注）命令語とアキュムレータとの間に1つ以上の空白を置いてもよく，図2.2の(1)と(2)，(3)と(4)は同じ意味です．

b．アセンブラ命令

アセンブラを制御する命令です．

図2.2　アキュムレータ番地指定の命令欄の記述例

2.5 オペランド欄

オペランド欄の解釈は，命令欄に依存します．オペランド欄が必要な場合には，命令欄に続けて１つ以上の空白を置いて指定します．簡略命令コードに対して，オペランド欄の番地指定形式（「4.2　番地指定」を参照）が指定されています．

オペランド欄の形式と対応する番地指定形式（アドレシングモード）は，表2.1の通りです．

アセンブラ命令のオペランド欄の形式は，簡略命令コードの場合の形式とは異なります．

表 2.1　オペランド形式と対応する番地指定形式

オペランド形式	番地指定形式
空オペランド	インヘレント
	アキュムレータ
＜式＞	直　　接
	拡　　張
	相　　対
#＜式＞	イミーディエイト
＜式＞，R	インデックス
＜＜式＞	直　　接
＞＜式＞	拡　　張
〔＜式＞〕	拡張間接
＜＜式＞，R	8ビットオフセットインデックス
＞＜式＞，R	16ビットオフセットインデックス
〔＜式＞，R〕	インデックス間接
＜〔＜式＞，R〕	8ビットオフセットインデックス間接
＞〔＜式＞，R〕	16ビットオフセットインデックス間接
Q＋	自動インクリメント（＋1）
Q＋＋	自動インクリメント（＋2）
〔Q＋＋〕	自動インクリメント間接
－Q	自動デクリメント（－1）
－－Q	自動デクリメント（－2）
〔－－Q〕	自動デクリメント間接
W_1〔，W_2，…Wn〕	イミーディエイト

記号の説明

R………PCR，S，U，X，Y レジスタのうち1つを表す.

Q………S，U，X，Y　　レジスタのうち1つを表す.

Wi（i＝1～n）………A，B，CC，D，DP，PC，S，U，X，Y レジスタのうちの1つを表す.

2.6　注　釈　欄

注釈欄は，その文についての注釈を記入する欄で，プログラマが任意に使用することができます．注釈欄は，オペランド欄の右側に，またはオペランド欄が空であるならば命令欄の右側に，1個以上の空白をおいて記入します．この欄では，どんなASCII 文字でも使用できます．

第3章　言語の基本規則

本章では，「第4章　機械命令」以降の説明で使用する基本的な規則について記述されています．例えば，文を書くのに必要な文字セット，アセンブラ言語の基本的な要素である項及び番地等です．

3.1 文字セット

アセンブラ言語で文を書く場合，表3.1に示す文字を使用することができます．

表3.1　文字セット

順番	種類	文字	名称	備考
1	英字	A～Z a～z	アルファベット	英字と数字をまとめて英数字という．
2	数字	0～9		
3	特殊文字	+	プラス	
		−	マイナス	
		*	アスタリスク	
		/	スラッシュ	
		(	左括弧	
		)	右括弧	
		,	コンマ	
		'	引用符	
		.	ピリオド	
		$	ドル記号	
		#	シャープ	
		@	単価記号	
		%	パーセント	
		&	アンパサンド	
		_	下線	
		;	セミコロン	
		:	コロン	
		!	感嘆符	
		[		
		]		
		<		
		>		
		^		
			空白（ブランク）	
4	その他	ASCIIコードから上記60文字を除いた文字すべて．		注釈欄，注釈文又は文字定数の中などに使用できる．

なお，制御用文字（文の終りを識別するための記号）として復帰（Carriage Return），改行（Line Feed）があります.

文字は，アセンブラ言語の最小の要素である項を構成し，項には記号，ロケーションカウンタ参照，自己規定項があります.

各々の項の記述上の規則は，本章「3.3 項」を，また英小文字の使用法については，「8.3 ソースプログラムファイルの作成」を参照してください.

3.2 番　地

ソースプログラムの機械命令文と一部のアセンブラ命令文は，ラベル欄に指定されている記号をオペランド欄に記入して参照します．このような対応は，アセンブラが出力する目的プログラムの中では，アセンブラがラベル欄の記号に与えた値（ロケーションカウンタの値という）で参照します.

番地の対応の例を図3.1に示します.

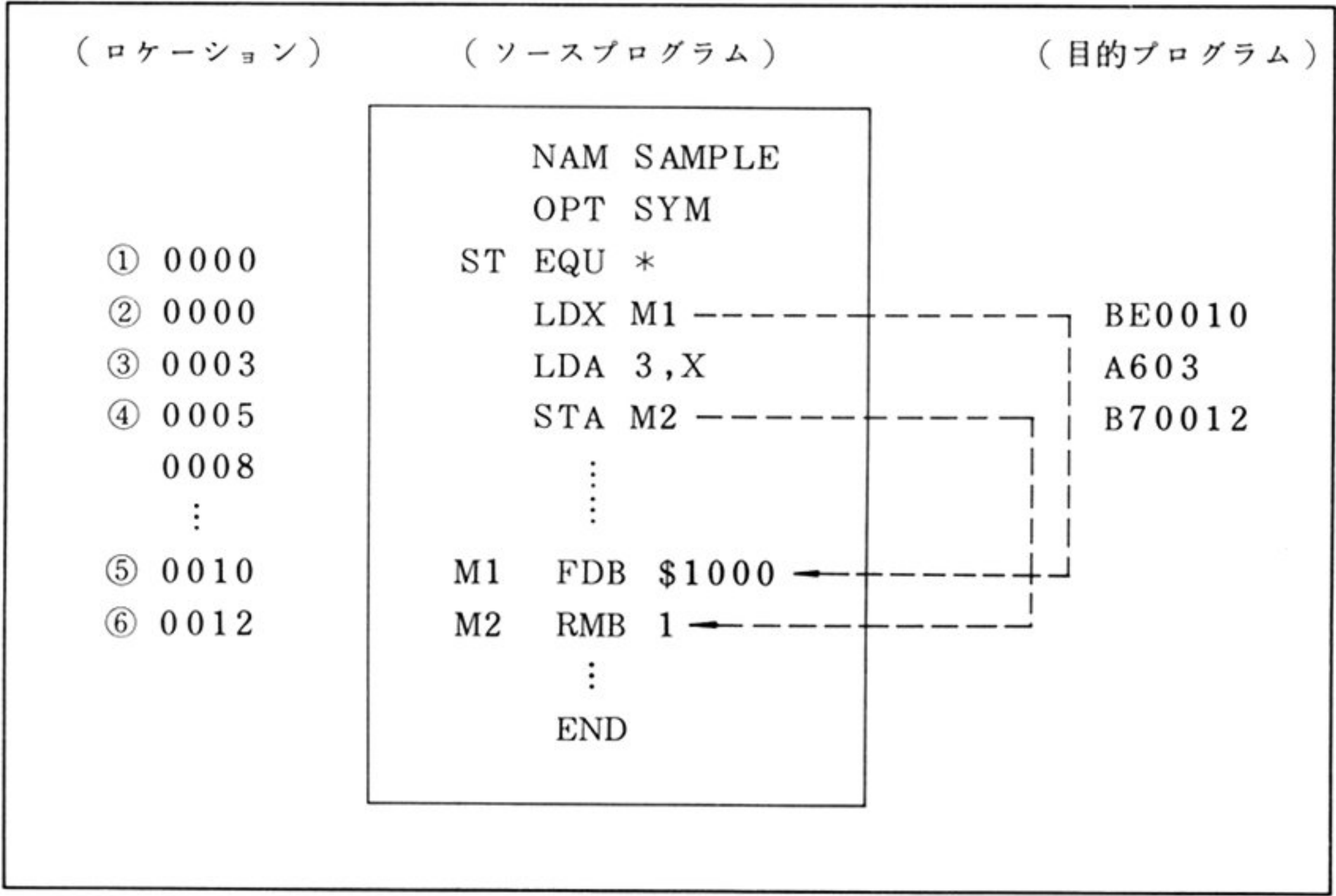



図3.1　番地の対応

図3.1でロケーションとは，プログラムが配置される位置を示すもので，命令文のこのような位置を明確にするために番地という言葉を使用します.

例えば，図3.1のM1及びM2の番地はそれぞれ$(10)_{16}$，$(12)_{16}$です．

ロケーションカウンタ（番地割当てカウンタ）とは，ソースプログラムの各々の命令文が持つ長さを計算して，各々の命令のロケーションを決定するものです．

ソースプログラムでは，オペランド欄でM1，M2を参照しますが，目的プログラムでは記号は使用されず，ロケーションの値（番地）で表現されます．

図3.1の①の行は，ラベルSTを定義しているだけで実行には何ら意味を持ちません．

図3.1の②の行は，1バイト目に簡略命令コードLDXの機械命令コードである$(BE)_{16}$が入り，2バイト目からはM1の番地$(0010)_{16}$が入って，LDXの拡張番地指定形式で目的プログラムに変換されたことを示します．④の行も②の行と同様に，STAの機械命令コード$(B7)_{16}$の後に，M2の番地$(0012)_{16}$が入り，目的プログラムに変換されたことを示します．

3.3 項

項は，値を表すアセンブラ言語の最小構成要素で式（「3.4 式」を参照）の中で用い，以下の3種類があります．

- 記　　号
- ロケーションカウンタ参照
- 自己規定項

3.3.1 記　　号

文のラベル欄に書かれたラベルを記号といい，他の文でそのラベルのついた文を参照できるようにするために使用します．

一般に記号は，値が割り当てられ，次の規則に従って作成します．

(a) 1～6文字の英数字及び特殊文字で構成します．

(b) 記号の中で使用できる文字は，英字（A～Z），数字（0～9），特殊文字．（ピリオド），$（ドル記号），_（下線）です．

(c) 第1文字は，A～Z又は．（ピリオド）で始めます．

(d) 特殊記号（A，B，CC，D，DP，PC，PCR，S，U，X，Y）は，アセンブラによって用いられるため，ラベル欄に書けません．

一つの記号は，ラベル欄に1度だけしか書くことはできません．もし重複して

定義されると，後から定義された記号は誤りとなります．

ラベル欄に書かれた記号（ラベルと呼ぶ）には通常，翻訳された命令又はデータの先頭バイトを示すプログラムロケーションカウンタの値が割り当てられます．

EQU 命令のラベル欄の記号には，オペランド欄に記述された式の値が与えられます．

また命令によっては，ラベル欄に名前を書いてはならないものがあり，ORG，NAM，END，OPT，TTL，PAGE，SETDP の命令が該当します．

記号の例を図 3.2 に示します．

```
正しい例
          JMP
          Z
          .A
          A$B

誤った例
          NOSYMBL     （規則(a)に違反）
          A/B         （規則(b)に違反）
          123         （規則(c)に違反）
          X           （規則(d)に違反）
```

図3.2　記号の例

3.3.2 ロケーションカウンタ参照

アセンブラは，ソースプログラムを翻訳するとき，記憶域の番地を割り当てるためにロケーションカウンタを用いて，各々の命令が占める大きさを累積計算します．計算されたロケーションカウンタの値をオペランド欄で，項として＊（アスタリスク）を書くことで参照することができます．機械命令及び定数を翻訳するとき，ロケーションカウンタは翻訳される命令や定数の先頭ロケーションをさす値を持ち，翻訳後，その命令や定数が占める記憶域の大きさ（バイト数）だけロケーションカウンタを増加させます．

ロケーションカウンタは，符号なし2バイト固定小数点データ（最大値 $(65535)_{10}$）として取り扱われ，アセンブラは，ロケーションカウンタの値が最大値を越えた場合には，誤りを表示します．

ロケーションカウンタの初期値は，ゼロに設定されますが ORG 命令で強制的にその値を変えることもできます.

ロケーションカウンタは，機械命令，FCB命令，FDB命令，BSZ命令，RMB命令及びEQU命令のオペランドで参照されます. 図3.3にロケーションカウンタの参照の例を示します.

```
(ロケーション)   (ソースプログラム)          (オペランド値)
                      ⋮
0100       S1   EQU     *                  0100
                      ⋮
0150            BNE     *+7                0157
0152            LDX     #$500
0155            BRA     *+5                015A
0157            LDX     #$600
015A            LDA     0,X
015C                  ⋮
```

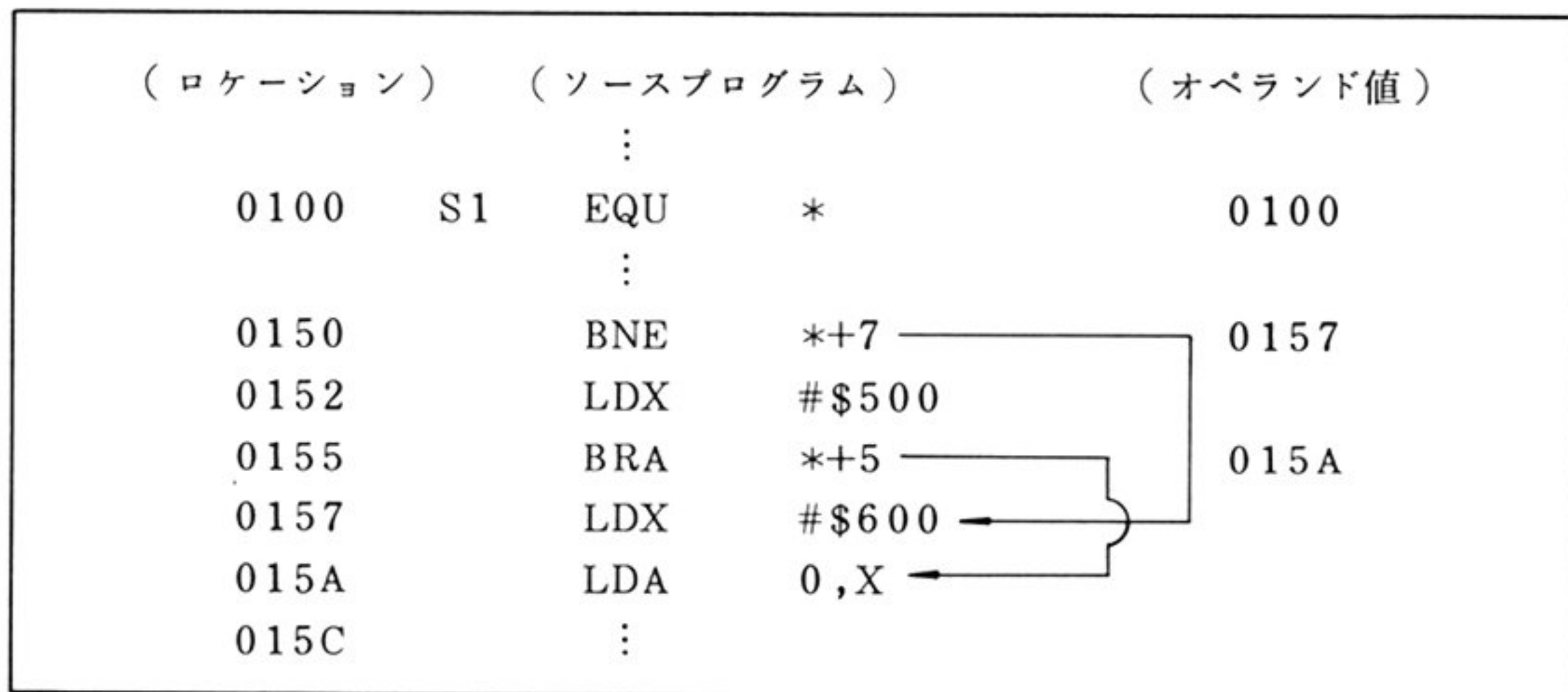



図3.3 ロケーションカウンタ参照の例

3.3.3 自己規定項

自己規定項とは，書かれたままの値を表現する一種の項です. 自己規定項を表現する方法として五つの形式が用意されています.

その形式及び記述形式を列挙します.

	自己規定項の形式	記述形式
(1)	2進自己規定項	%2進数
		2進数B
(2)	8進自己規定項	@8進数
		8進数O
		8進数Q
(3)	10進自己規定項	10進数
		&10進数
(4)	16進自己規定項	$16進数
		16進数H
(5)	文字自己規定項	'文字

ここで２進，８進，１０進及び１６進の各数は表3.２に示す文字で構成された一つ又はいくつかの文字列です．また，数値表現の対応を表3.３に示します．

表3.2　表現する数の構成文字

表現する数	構成文字
2進数	01
8進数	01234567
10進数	0123456789
16進数	0123456789 ABCDEF

表3.3　数値表現の対応

2進	8進	10進	16進
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

アセンブラは，これらの数値，或いは文字が表す値を１６ビットの２進数に変換します．変換する途中で数値の大きさが１６ビットを越えた場合には上位のビットが失われ，その後，変換された値を命令の該当部分に，命令の長さに従って格納されます．

以下自己規定項を形式別に説明します．

(1)　２進自己規定項

２進自己規定項は，文字％に続けて２進数を書くか，又は２進数を書き最後に文字Ｂを書くことにより表します．図3.４に例を示します．

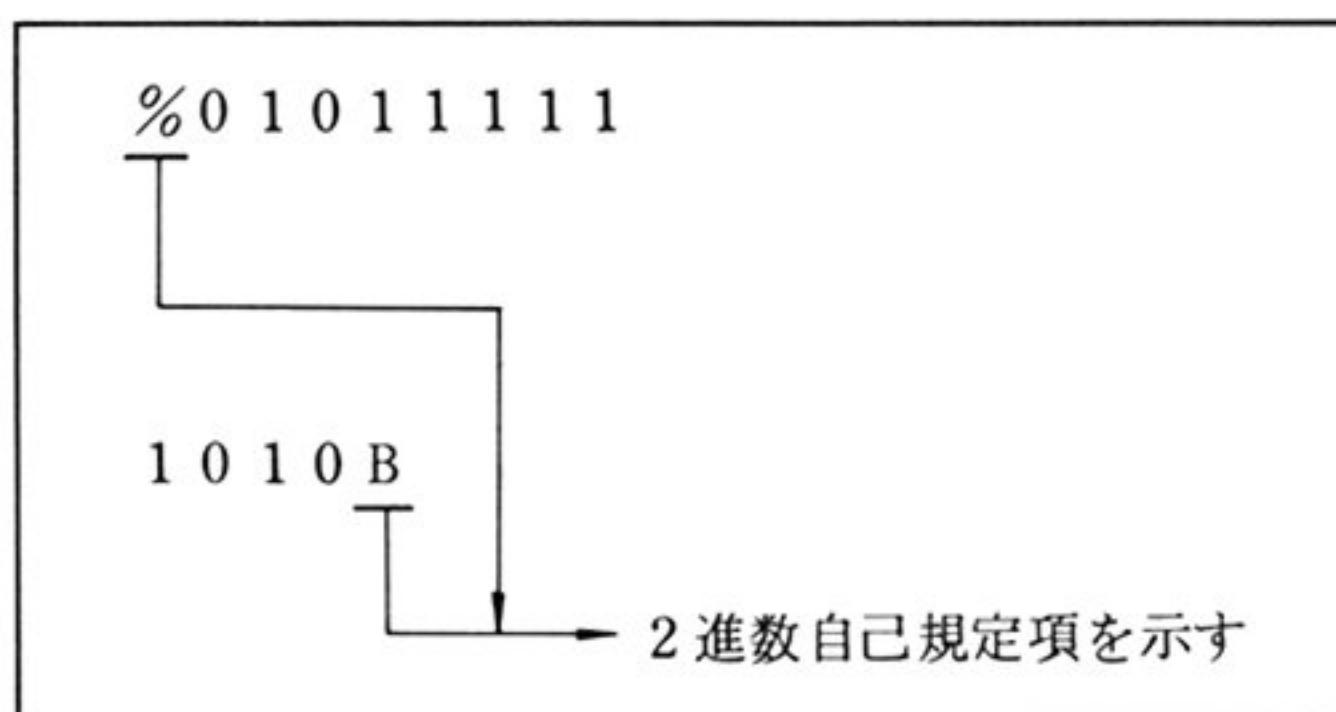



図3.4　2進数自己規定項の例

(2)　8進数自己規定項

8進数自己規定項は，文字@に続けて8進数を書くか，又は8進数を書き最後に文字OかもしくはQを書くことにより表します．図3.5に例を示します．

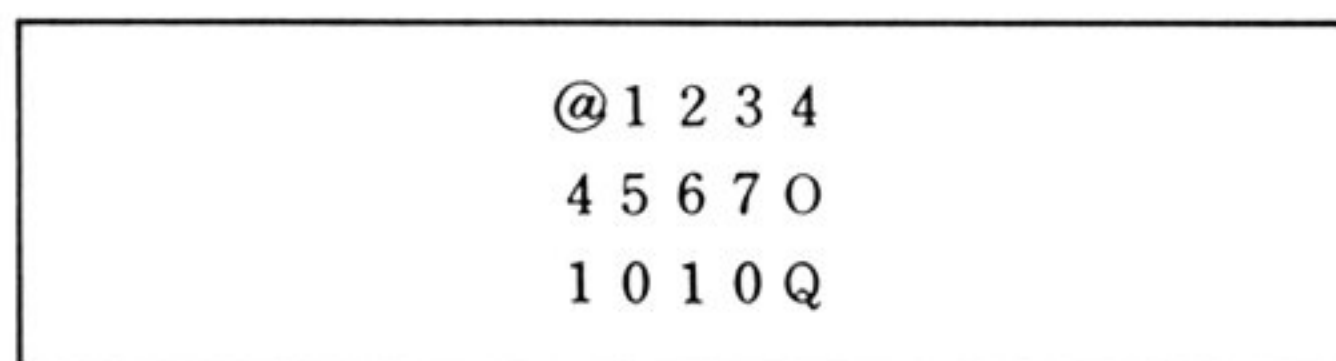



図3.5　8進数自己規定項の例

(3)　10進数自己規定項

10進数自己規定項は，文字&に続けて10進数を書くか，又はそのまま符号なし10進数を書くことにより表します．図3.6に例を示します．

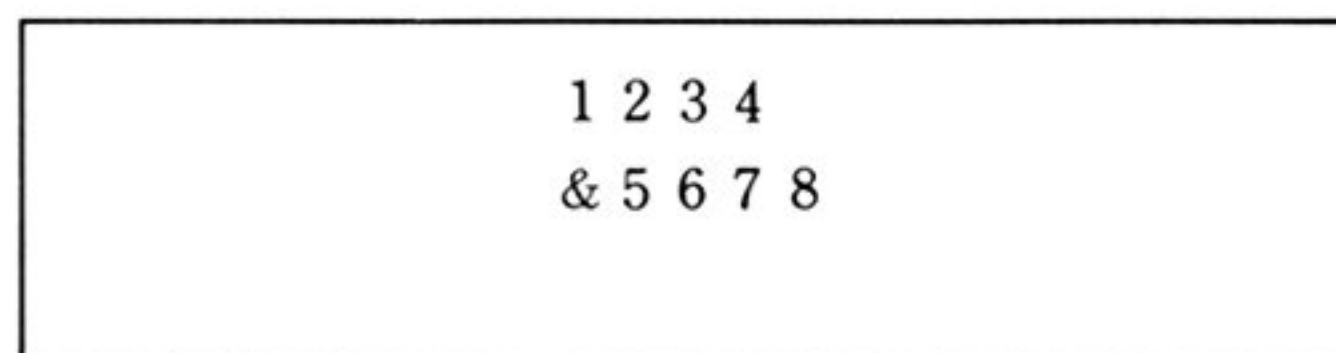



図3.6　10進数自己規定項の例

(4)　16進数自己規定項

16進数自己規定項は文字$に続けて16進数を書くか，又は16進数を書き最後にHを書くことにより表します．文字Hを使って表現する16進数は，その左端の文字が数字以外の文字であってはいけません（英字の場合は，記号と見なされる）．図3.7に例を示します．

```
$FFFF
0FFFFH
1234H
```

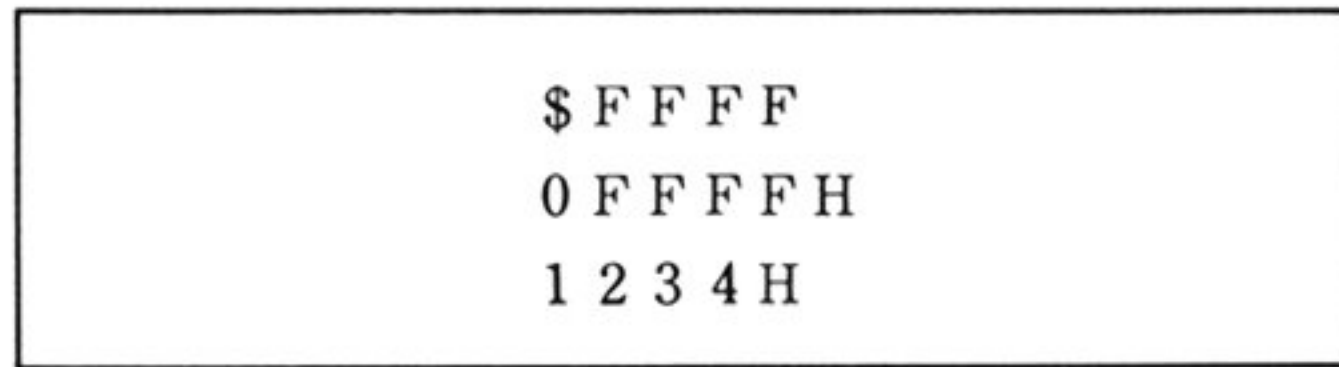



図3.7　16進数自己規定項の例

(5)　文字自己規定項

文字自己規定項は，文字コードを数値として表現するために用いられ式の中に書くことができます．

文字自己規定項に使用できる文字は，ASCIIであり，アセンブラは，文字定数を2進の文字コードに変換し，命令によって定められている定数領域に組み込みます．

文字自己規定項は，'（アポストロフィ）を使用して記述します．図3.8に例を示します．

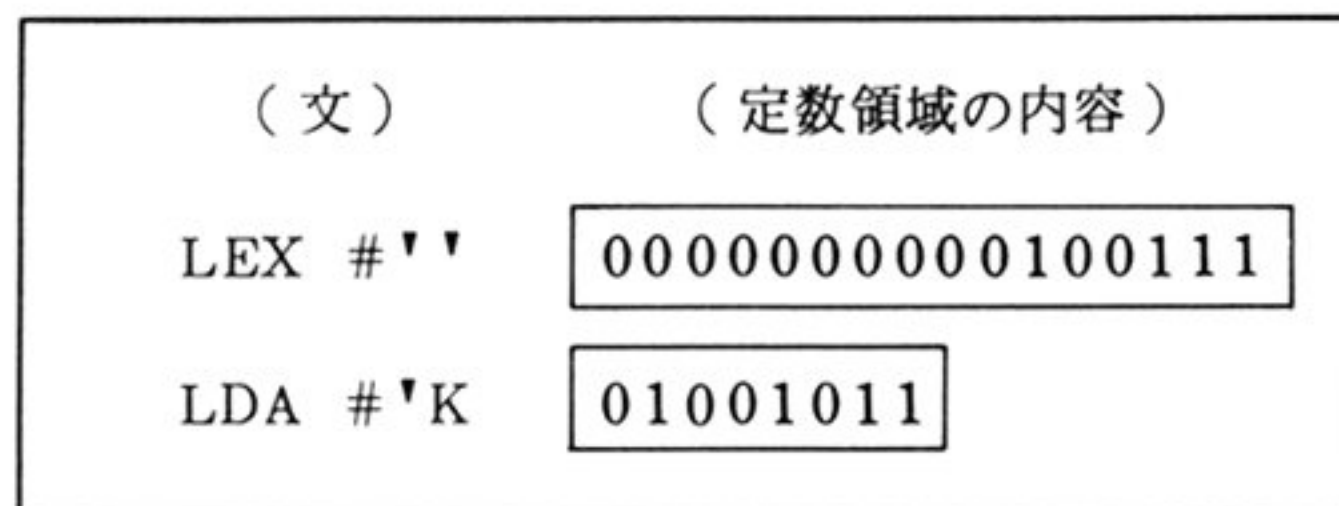



図3.8　文字自己規定項の例

3.4　式

アセンブラ言語は，命令文のオペランド欄に式を書くことができます．具体的には番地を表す部分，RMB命令やBSZ命令の領域の大きさを表す部分などに使用します．例を図3.9に示します．

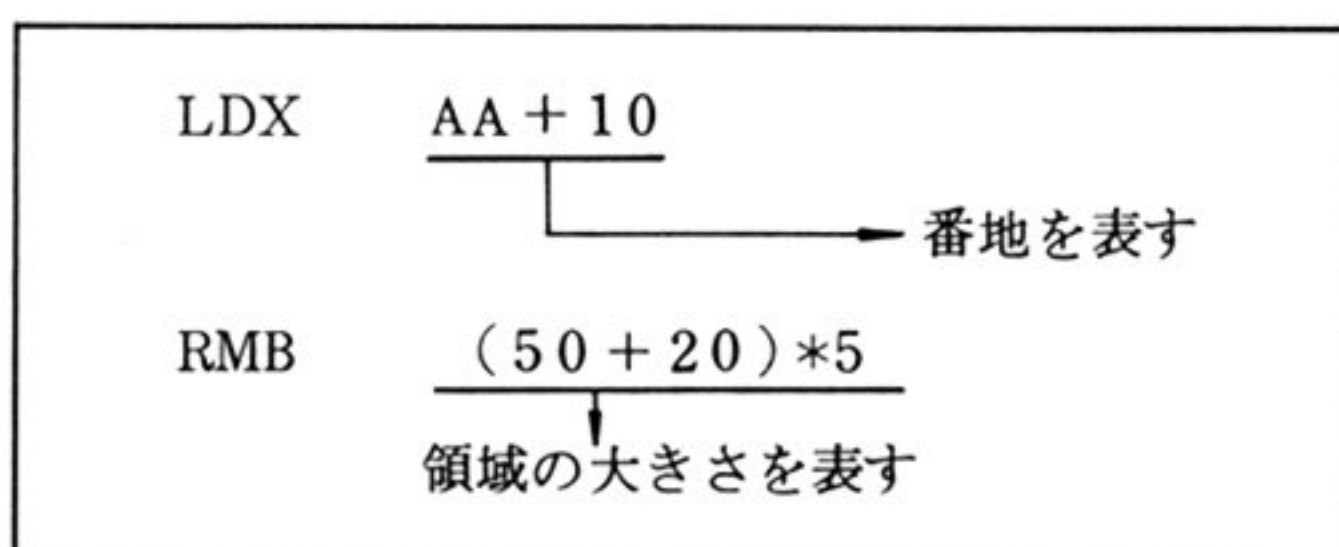



図3.9　オペランド欄の式の例

式は，記号，自己規定項，演算子及び括弧を組み合せたものであり，機械命令あるいはアセンブラ命令のオペランドとして用いられる数値を表します．

3.4.1 式に関する規則

式を記述するときの規則を次に示します．

(1) 式は一つの項又はいくつかの項を一般的な代数規則に従って演算子と括弧を使って結合したものから構成されます．

(2) 単項演算子ユーナリマイナスは，式の最初にあってもよいが他の演算子は，式の最初に記述することはできません．

(3) 式の中で，項又は演算子を連続して用いてはなりません．

3.4.2 2文字演算子

加減乗除の演算子に加えて，2文字演算子を用いることができ，各々の2文字演算子は，感嘆符（！）で始まり二つのオペランドを必要とします．

次に2文字演算子の定義を表3.4に示します．

表3.4　2文字演算子の定義（つづく）

2文字演算子	名　称	定　義
！∧	巾乗	左オペランドが右オペランドで指定された巾だけ累乗される．もし，右オペランドが0ならば左オペランドの値にかかわらず結果は1となる．
！・	論理積	左オペランドの各ビットと右オペランドの対応するビットの論理積が求められる．
！＋	論理和	左オペランドの各ビットと右オペランドの対応するビットの論理和が求められる．
！×	排他的論理和	左オペランドの各ビットと右オペランドの対応するビットの排他的論理和が求められる．
！＜	左シフト	左オペランドを右オペランドで指定されたビット数だけ左シフトする．左オペランドは右側からゼロが補充される．
！＞	右シフト	左オペランドを右オペランドで指定されたビット数だけ右シフトする．左オペランドは左側からゼロが補充される．

表 3.4　2文字演算子の定義（つづき）

2文字演算子	名　称	定　義
！L	左回転	左オペランドを右オペランドで指定されたビット数だけ左回転する．左オペランドの最上位ビットは回転されて，最下位ビットの位置へ移る．
！R	右回転	左オペランドを右オペランドで指定されたビット数だけ右回転する．左オペランドの最下位ビットは回転されて，最上位ビットの位置へ移る．

3.4.3 式の値の決定

式の値は，次の規則で計算されます．

⑴ 各項ごとにその値が求められます．

⑵ 式の値は，同じ優先順位を持つ演算子は，左から右へと評価されます．演算子の優先順位は以下のとおりです．

第1位　（，）………括弧

第2位　＊，／，！∧，！・，！＋，！×，！<，！>，！L，！R…乗除算，2文字演算

第3位　＋，－………加減算（ユーナリマイナスを含む）

⑶ 式の値は，16ビットの2進数として計算され，また負の値は2の補数形式で取り扱われます．したがって16ビットを越える数値は，上位部分が無視されます．また式の演算途中の値が，16ビットを越えた場合も同様です．

⑷ 式の演算途中及び演算結果の値は，整数値として評価します．

⑸ 除数が0の除算も許されており結果は$(65535)_{10}$になります．

3.5 簡略命令コード

命令において動詞に相当する部分で，文の動作を規定するものであり命令欄に書きます．

アセンブラ言語では，プログラマが容易に命令コードを覚えることができるように，すべての命令コードは，記号化された簡略命令コードで表現されます．

簡略命令コードは次の2種類に大別されます．

(1) 簡略機械命令コード

簡略機械命令コードは，マイクロプロセッサMBL6809 の機械命令に対応する簡略コード（「付録1　機械命令一覧表」参照）からなります．

(2) アセンブラ命令コード

アセンブラ命令コードとは，アセンブラにより実行される補助機能を指定するための命令コードであり，機械命令を作成することはありません．
（「付録3　アセンブラ命令一覧表」参照）

第4章　機械命令

本章では，機械命令の形式，記述上の規則等を説明します．個々の機械命令の機能については別冊のユーザマニュアル「MBL6809・MBL6809E」及び「付録1 機械命令一覧表」を参照してください．

機械命令はすべて記号化されており，アセンブラは翻訳時に，この記号化された機械命令をマイクロプロセッサMBL6809が実行する目的プログラムに翻訳します．

機械命令は，プログラムの実行時に指定されたオペレーションを実行できる命令です．アセンブラが翻訳時に処理するFCB命令，FDB命令，RMB命令等は機械命令でなく，実行のために必要な定数とか，領域を目的プログラムに確保するアセンブラ命令です．

4.1　機械命令の形式

機械命令文の一般的な記述形式及び記述上の規則は，次のとおりです．

〔形　式〕

ラベル欄	命　令　欄	オ　ペ　ラ　ン　ド
〔記号〕	簡略命令コード 〔{A B}〕	〔{A B 式 〔〔式〕〕，R #式 〔〔式 or Acc，〕R〕}〕

（注） R……X，Y，S，U，PCRレジスタのうちの1つを表す．
Acc……A，B，Dレジスタのうちの1つを表す．

〔規　則〕

(1)　ラベル欄に記号を記述した場合，その記号には機械命令の先頭番地が与えら

れます.

⑵ オペランド欄のレジスタ記号A又はBは，命令欄の簡略命令コードに続けて記述することができます.

⑶ オペランド欄に，'式，R'の形式で記述する場合，式の値が0ならば，式又は式とコンマ(,)を省略することができます.

⑷ 同様に，'〔式，R〕'の形式で記述する場合，式の値が0ならば，式又は式とコンマ(,)を省略することができます.

4.2 番 地 指 定

機械命令は，演算の対象となるデータを参照あるいは格納したり，実行制御を移行（分岐）したりするためのメモリ領域の番地，又はレジスタの指定を行なう番地指定（アドレッシング）が必要です.

機械命令の番地指定には次の8種類があります.

- インヘレント番地指定
- アキュムレータ番地指定
- イミーディエイト番地指定
- インデックス番地指定
- 相対番地指定
- 直接番地指定
- 拡張番地指定
- レジスタ番地指定

4.2.1 インヘレント番地指定

簡略命令コード自体に処理の対象となる情報が含まれている番地指定で，オペランド欄には何も記入しません.

マイクロプロセッサ（MPU）の内部レジスタ類が，簡略命令コードにより指定されます.

この番地指定の命令は，翻訳の結果1バイト又は2バイト（SWI 2，SWI 3）の機械命令コードになります．処理形態を図4.1に示します.

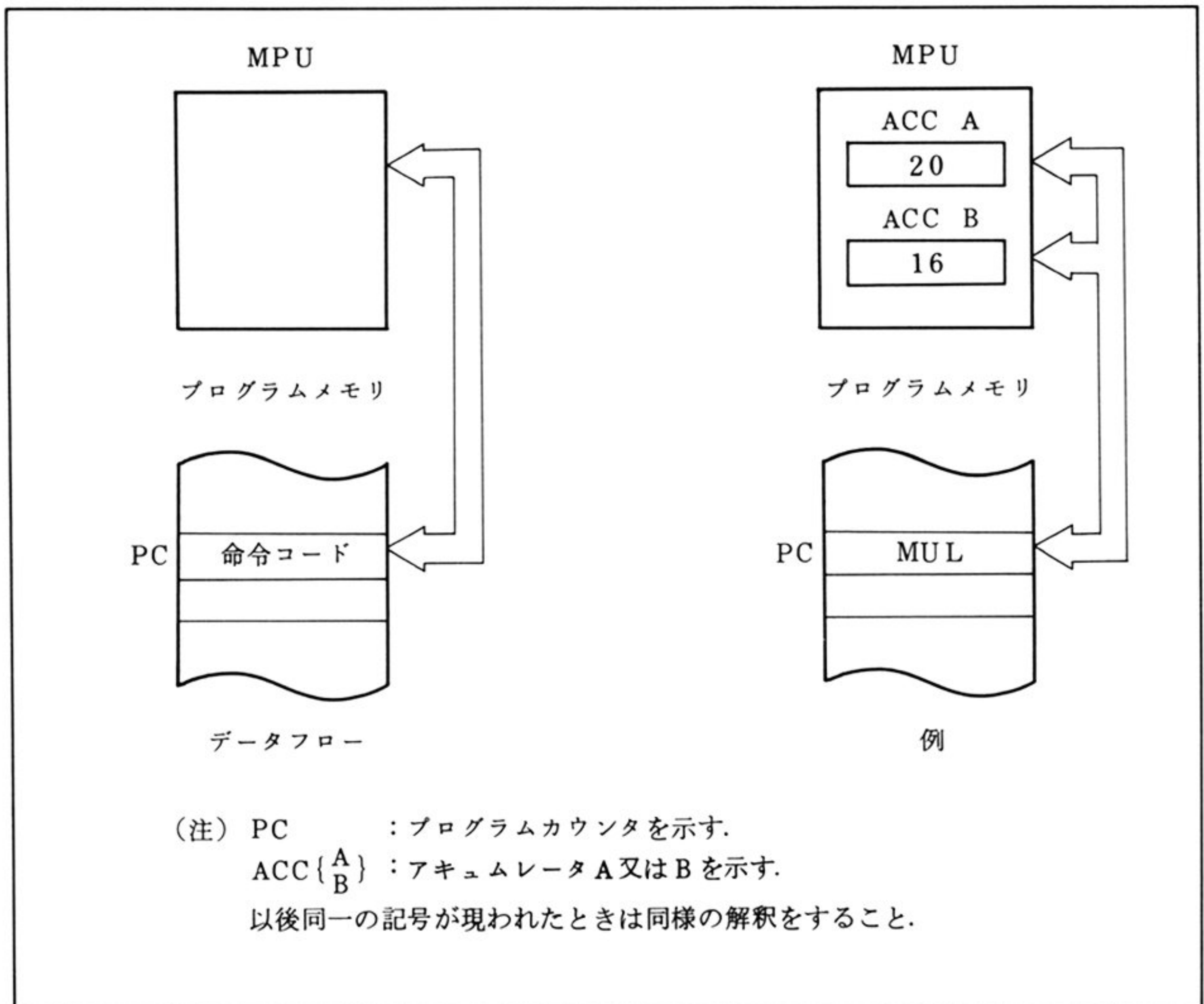



図 4.1　インヘレント番地指定の処理形態

4.2.2 アキュムレータ番地指定

アキュムレータA又はBのいずれかを指定する形式です．命令のオペランド欄にA又はBの文字を記述することにより行なわれます．

この番地指定では，簡略命令コードとオペランドA又はBとの間の空白は省略できます．また，この形式は翻訳の結果1バイトの機械命令コードになります．処理形態を図4.2に示します．

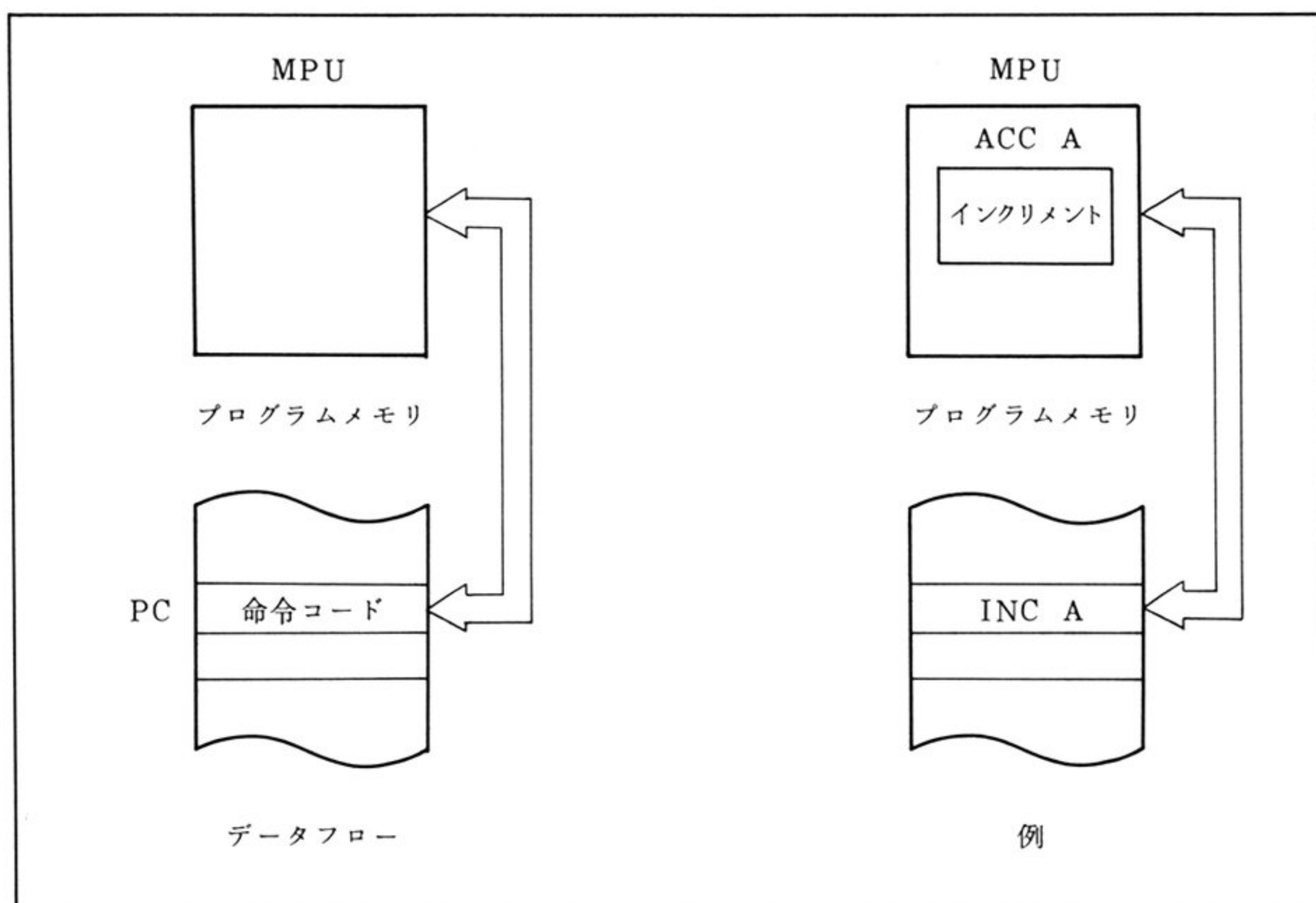



図 4.2 アキュムレータ番地指定の処理形態

4.2.3 イミーディエイト番地指定

イミーディエイト番地指定は，命令のオペランドの値を直接データとして扱う番地指定です.

この番地指定形式は．命令によって決まっている場合と，オペランド欄の記述を#（シャープ）で始めることによって指定される場合とがあり，オペランド欄には式が記入でき，翻訳の結果，機械命令の番地部に1バイトもしくは2バイトの大きさのデータとして組み込まれます.

番地物の大きさは命令によって異なり，1バイト又は2バイトです．番地部の大きさが1バイトの場合，−128から255 の範囲の値を持つ式を記述することができ，また番地部の大きさが2バイトの場合には，一般の式を記述することができます.

イミーディエイト番地指定の処理形態と記述例を図4.3と図4.4にそれぞれ示します.

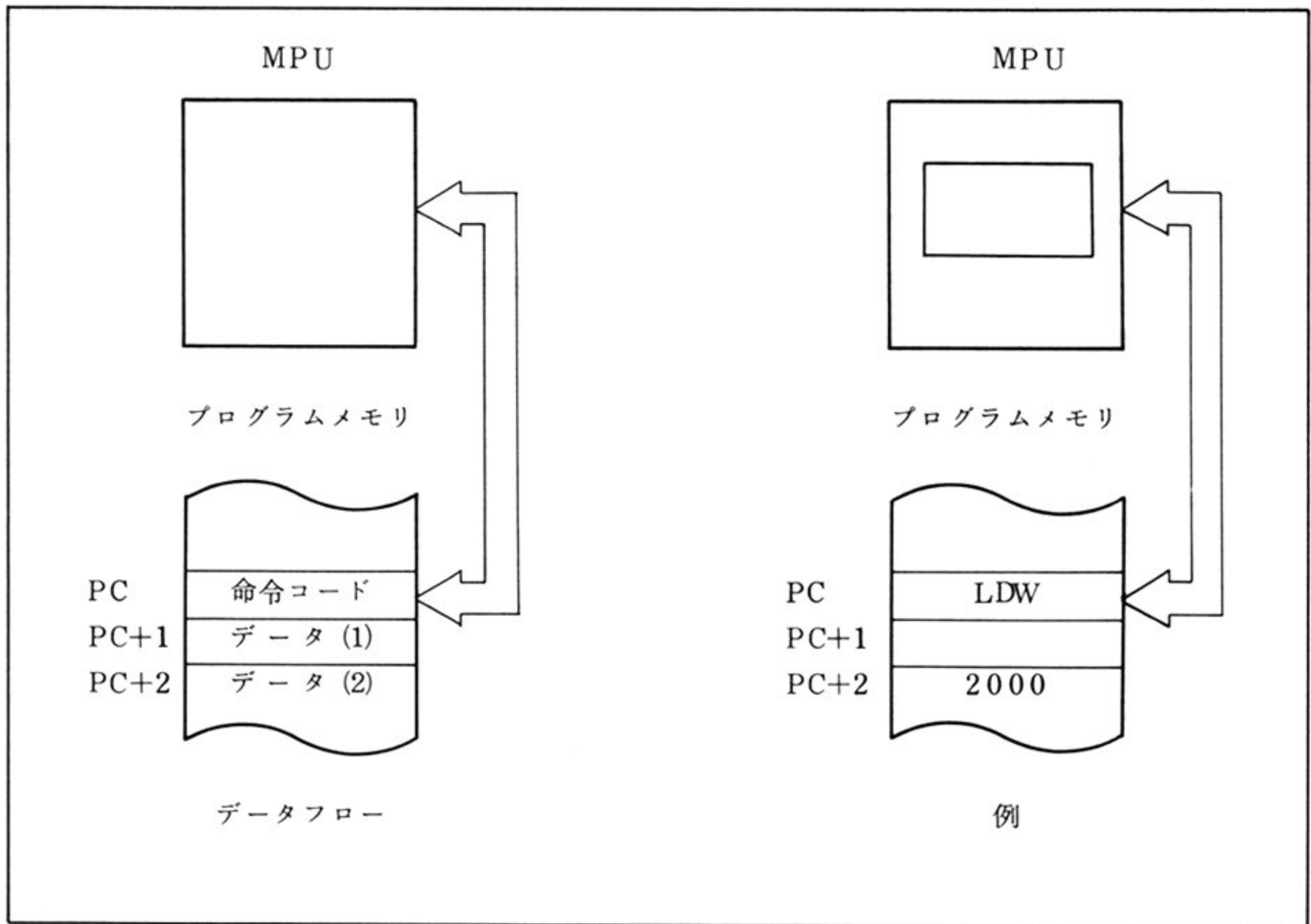



図 4.3　イミーディエイト番地指定の処理形態

LDA	#−1	（番地部には１６進でＦＦの値が入る）
LDX	#−1	（番地部には１６進でＦＦＦＦの値が設定される）
LDX	#65535	（同　上）
CMPB	#'A	（アキュムレータＢが１６進の４１と比較される）

図 4.4　イミーディエイト番地指定の記述例

4.2.4　インデックス番地指定

インデックス番地指定は，１つのレジスタに対して相対的であり，オペランド欄に次のように記述することによって指定されます．

式，R　　又は　　〔式，R〕

（注）R：X，Y，S，Uレジスタ及びPCRレジスタのうちの一つを示す．

前者の記述形式で番地は，命令実行時にレジスタの現在の内容の値を加算する

ことにより求められます.

後者の記述形式(間接形式)では番地は,命令実行時にレジスタの現在の内容と式の値を加算して求めた番地と,その番地の直後の番地から2バイトの値を取り出すことにより求められます.

インデックス番地指定として,次のような形式があります.

- 式オフセット形式
- アキュムレータオフセット形式
- 自動インクリメント
- 自動デクリメント
- プログラムカウンタ相対形式

(1) 式オフセット形式

式オフセット形式の一般形は次のように表されます.

R	〔R〕
,R	〔,R〕
式,R	〔式,R〕

(注) R:X,Y,S,Uレジスタのうち一つを示す.

式が指定されないかあるいは式の値が0であるとオペランド(番地部)はポストバイト(「付録2 ポストバイトの形式」参照)のみが生成されます.

間接形式でなく式の値が−16から+15の範囲(ただし,0は除く)であると,オペランドはレジスタ指定と式の値を含んだポストバイトのみが生成されます.実行時に式の値は,符号付き16ビットに拡張されてからレジスタに加算されます.

その他の一般形は,ポストバイトとともに式の値を含んだ1又は2バイトのオフセットを生成し,オフセットの大きさは式の型及び大きさによって決定されます.

−128から+127の範囲の値を持つ式は,8ビットオフセットを生成します.ただし,式が前方参照記号(定義される以前に参照されている記号)を含んでいると,フェーズエラー(パス1とパス2でのロケーションカウンタの不一致)を回避するために,16ビットオフセットを用いて翻訳されます.

8ビットオフセットが生成される場合には,その値は実行時に符号付き16ビットに拡張されます.

他の全ての場合は，16ビットオフセットが生成されます．

また，式オフセット形式では前述の一般形の先頭に“＜”又は“＞”を付けて，8又は16ビットオフセットを強制することができます．

記述形式は，次のとおりです．

＜R	＜〔R	＞R	＞〔R
＜，R	＜〔，R	＞，R	＞〔，R〕
＜式，R	＜〔式，R〕	＞式，R	＞〔式，R〕

記号“＜”は，8ビットオフセットを生成することを強制することができますが，式が−128から＋127の範囲の値を持つ式でないと，バイトオーバフローエラーが生成されます．

記号“＞”は，16ビットオフセットを生成することを強制する意味を持っています．

式オフセット形式及び式オフセット（間接）形式の処理形態を図4.5と図4.6に示します．

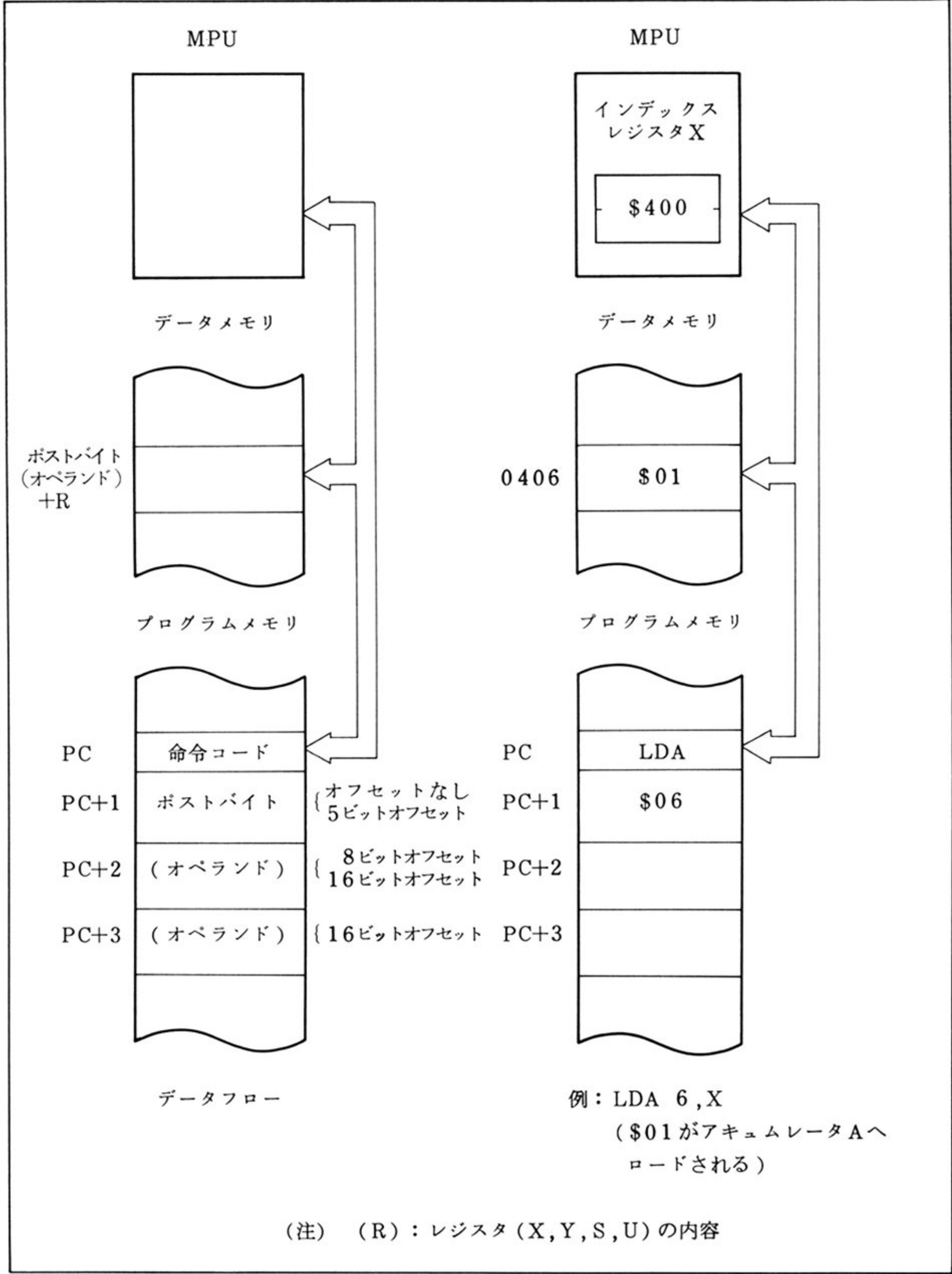



図4.5　式オフセット形式の処理形態

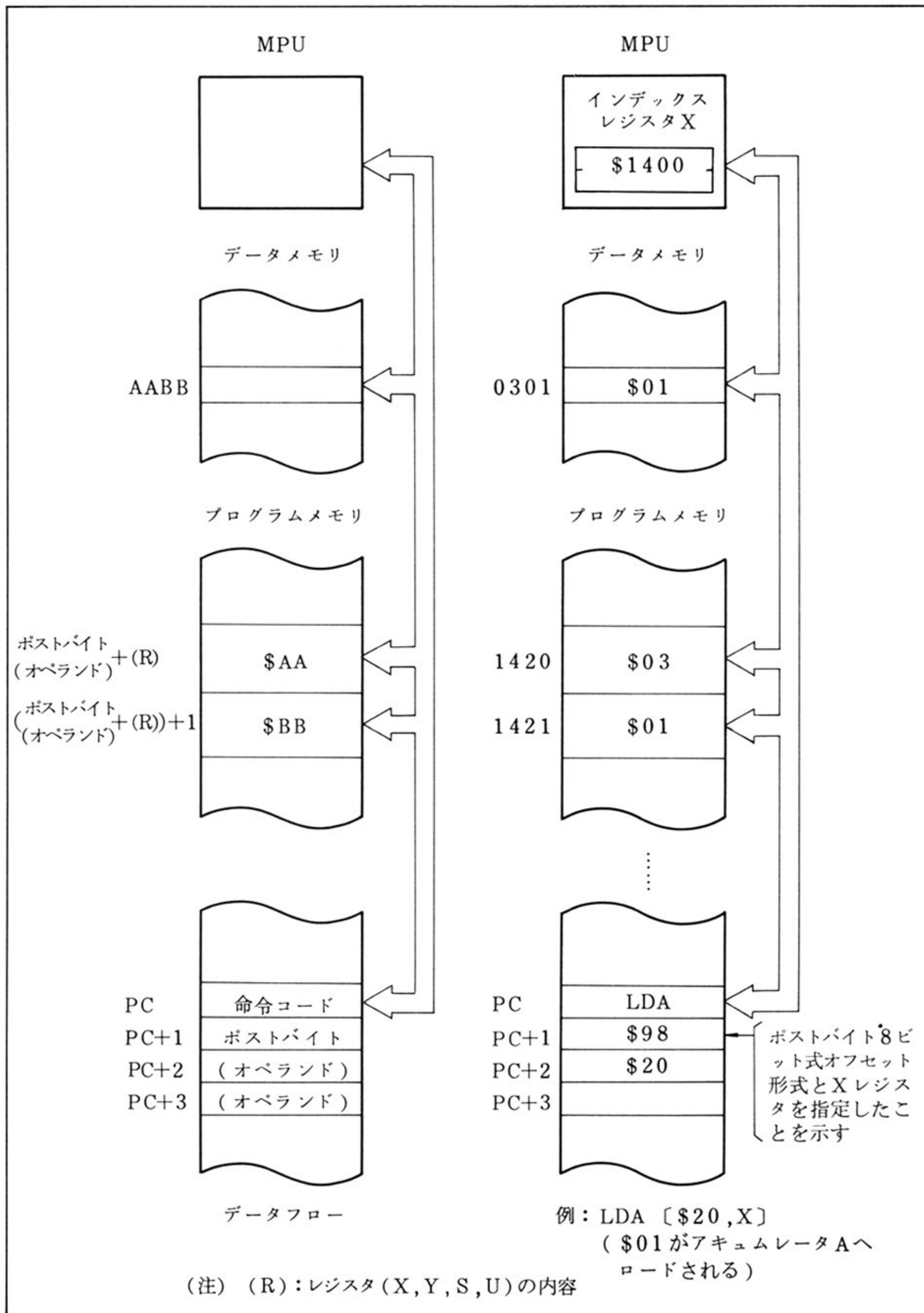



図4.6　式オフセット（間接）形式の処理形態

(2) アキュムレータオフセット形式

オペランド欄に，式の代りにアキュムレータ（A，B，D）を指定できます．この形式をアキュムレータオフセット形式といいます．

一般形は，

Acc,R　　又は　　〔Acc,R〕

(注) Acc：アキュムレータA，B，Dのうちの一つを示す．
　　R　：X，Y，S，Uレジスタのうち一つを示す．

で表されます．

この形式は，ポストバイトのみのオペランドを生成します．

アキュムレータオフセット形式の処理形態を図4.7に示します．

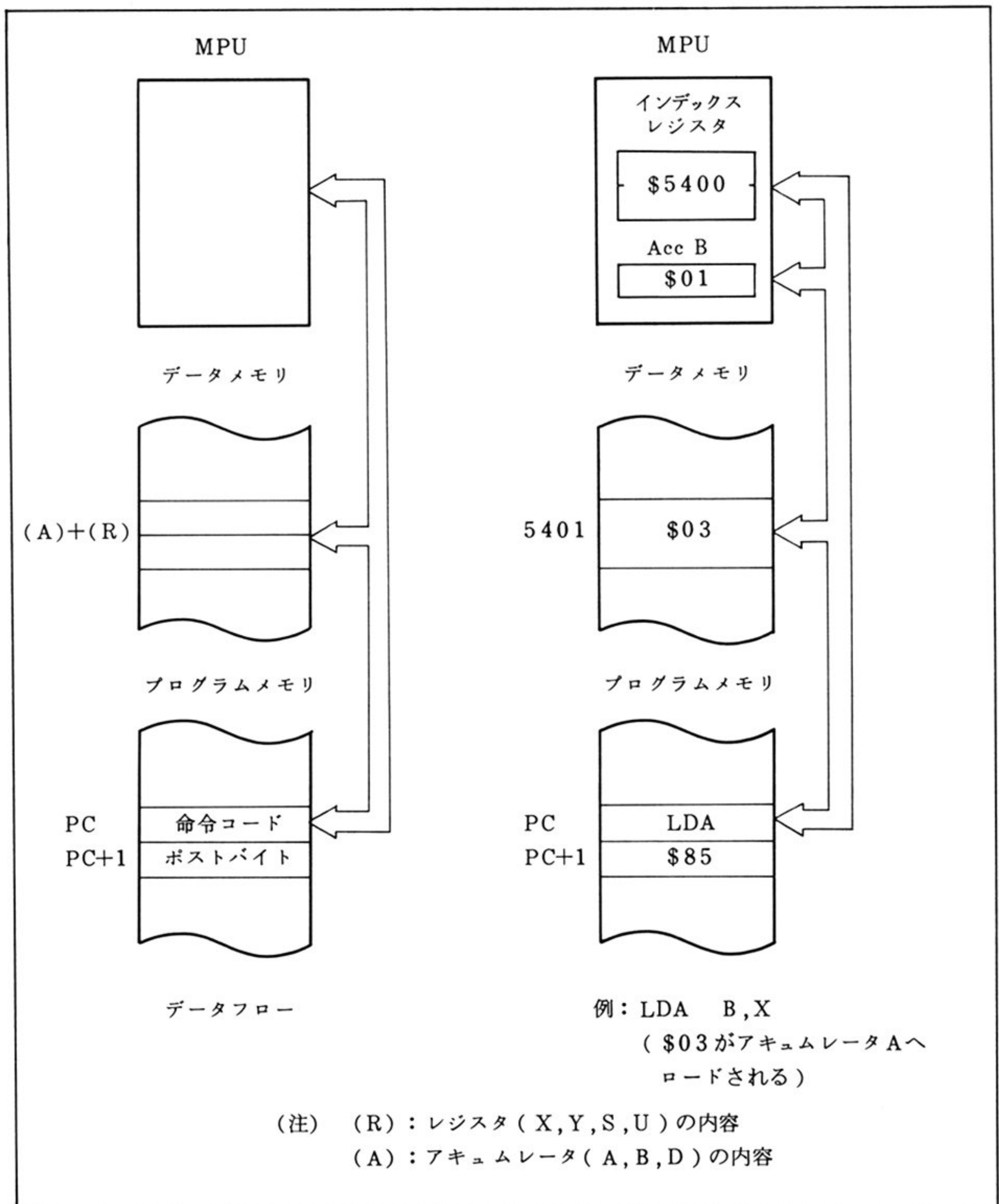



図 4.7　アキュムレータオフセット形式の処理形態

(3) 自動インクリメント及び自動デクリメント

自動インクリメント及び自動デクリメントは次の形式で表されます.

〔自動インクリメント〕

R+	,R+	0,R+
R++	,R++	0,R++
〔R++〕	〔,R++〕	〔0,R++〕

〔自動デクリメント〕

−R	,−R	0,−R
−−R	,−−R	0,−−R
〔−−R〕	〔,−−R〕	〔0,−−R〕

(注) R:X,Y,S,Uレジスタのうちの一つを示す.
- 各行の三つの項は同等である.
- 許される式の値は0だけである.

自動インクリメントは, 命令実行後に指定したレジスタに1又は2を加えます.

自動デクリメントは, 命令実行前に指定したレジスタから1又は2を引きます.

二つの形式は, ポストバイトのみのオペランドを生成します. 自動インクリメント及び自動デクリメントの処理形態を図4.8と図4.9に示します.

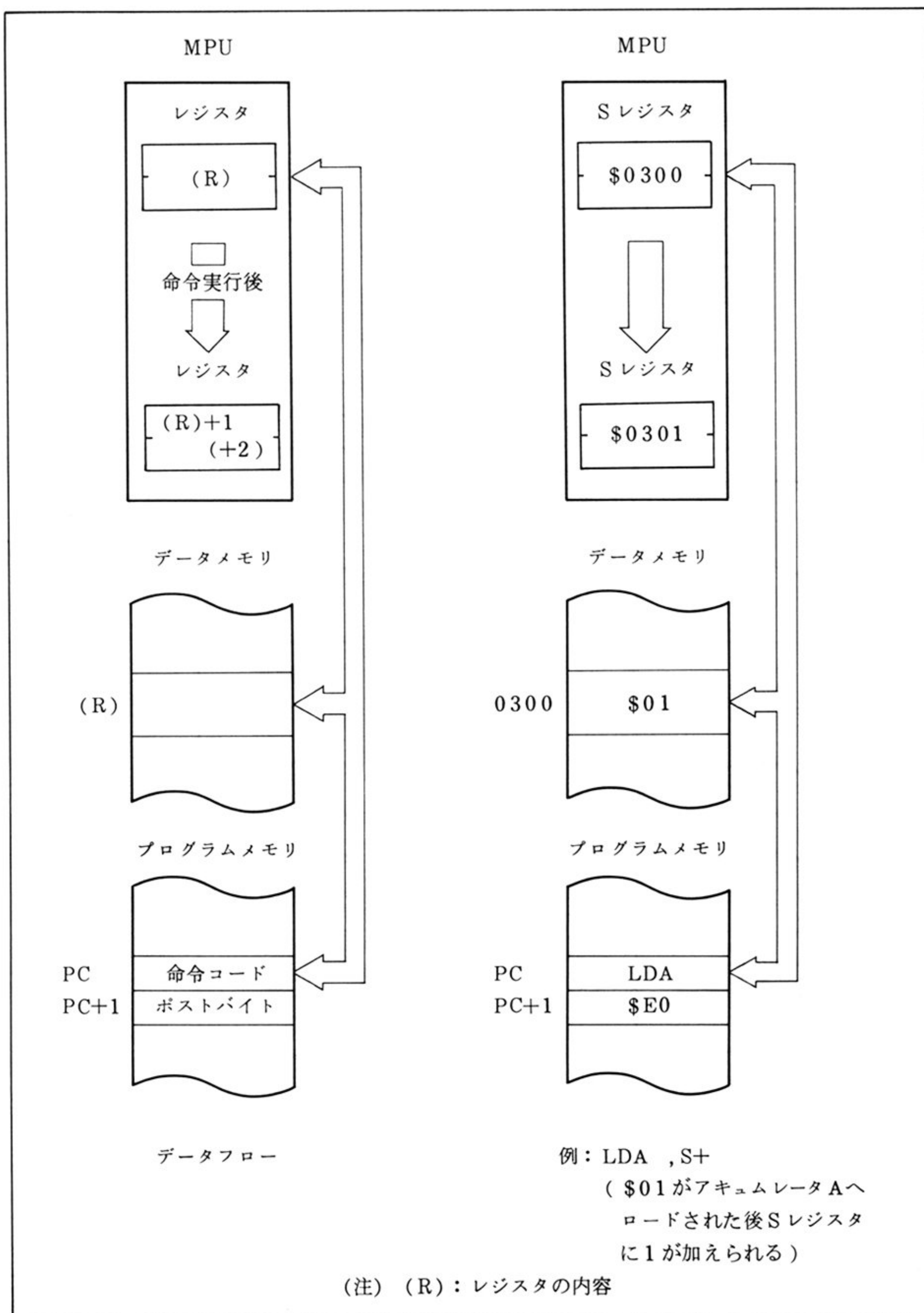



図 4.8 自動インクリメントの処理形態

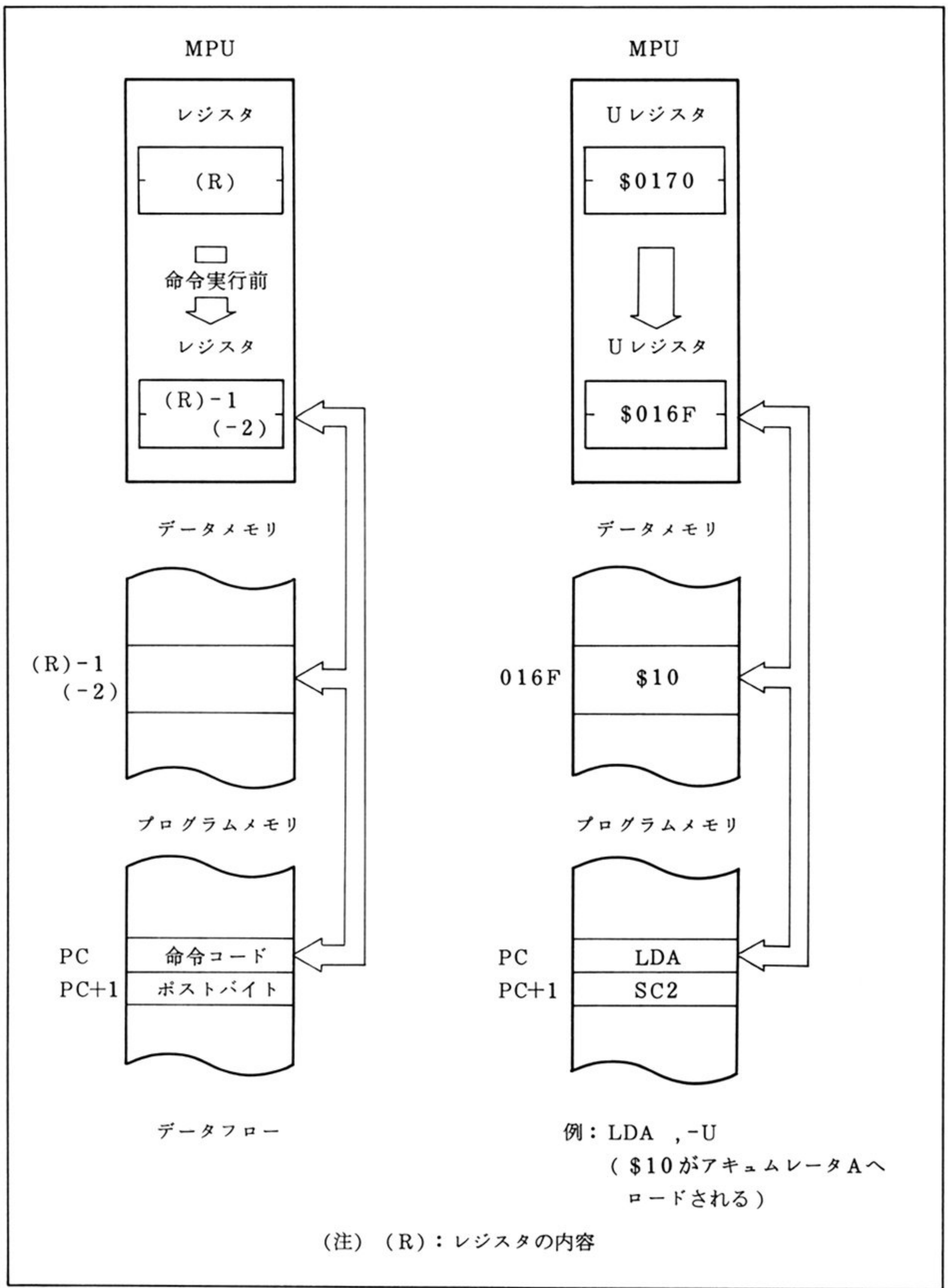



図4.9 自動デクリメントの処理形態

(4) プログラムカウンタ相対形式

プログラムカウンタ相対形式は，レジスタ指定としてPCRレジスタのみを記述する形式です．

一般形は，

式，PCR　　又は　〔式，PCR〕

で表されます．

この形式は，レジスタ指定が常にPCRレジスタであることとオフセットの生成法が異なるほかは，式オフセットと同じ処理形態です．（ただし，この形態では8／16ビットオフセットしかない）．

プログラムカウンタ相対形式では，アセンブラが生成する相対番地をポストバイトに続く8又は16ビットのオフセットとして用います．

(注)　相対番地：インデックス命令の直後のロケーションと式の値と差を2の補数で表したもの．

計算された相対番地が－128から＋127の範囲外か，あるいは式が前方参照記号を含んでいると，16ビットのオフセットとして用いられます．

式オフセット形式同様，オペランドの先頭に“＜”又は“＞”を付けることによって8又は16ビットオフセットを強制することができます．

8ビットオフセットを強制したときは，相対番地が－128から＋127の範囲にないと，バイトオーバーフローエラーが生成されます．

インデックス番地指定の記述例を図4.10に示します．

```
〔正しい例〕
    LDA     256,X
    LDA     A,Y
    STB     〔6,PCR〕
    ADDB    ,X+
〔誤った例〕
    LDA     〔,X+〕     自動インクリメント(+1)は間接形式で
                        は使えない.
    LDB     2,-U       自動デクリメントで許される式の値は0で
                        ある.
    STB     B,PCR      アキュムレータオフセット形式でPCRレジ
                        スタは指定できない.
```

図4.10　インデックス番地指定の記述例

4.2.5 相対番地指定

相対番地指定は，分岐命令によって用いられオペランド欄には，番地を示すための式を記述します.

翻訳の結果，ショート分岐命令では1バイト，ロング分岐命令では2バイトのオフセットが計算されて，機械命令の番地部に組み込まれます.

オフセットとは，分岐命令の直後のロケーション（例えばショート分岐命令なら，現在のプログラムカウンタに2を加えた値）と分岐先のロケーション間の差を2の補数形式で表わしたものです.

ショート分岐命令では，機械命令コードは1バイトで番地部は1バイトのオフセット値が組み込まれますので，式の値は－126から＋129の範囲でなければなりません.

ロング分岐命令では，機械命令コードは1又は2バイトで番地部の2バイトのオフセット値が組み込まれますので，式の値は0000からFFFF（16進）の全アドレス指定範囲で指定できます.

次に，相対番地指定の処理形態を図4.11と図4.12，記述例を図4.13に示します.

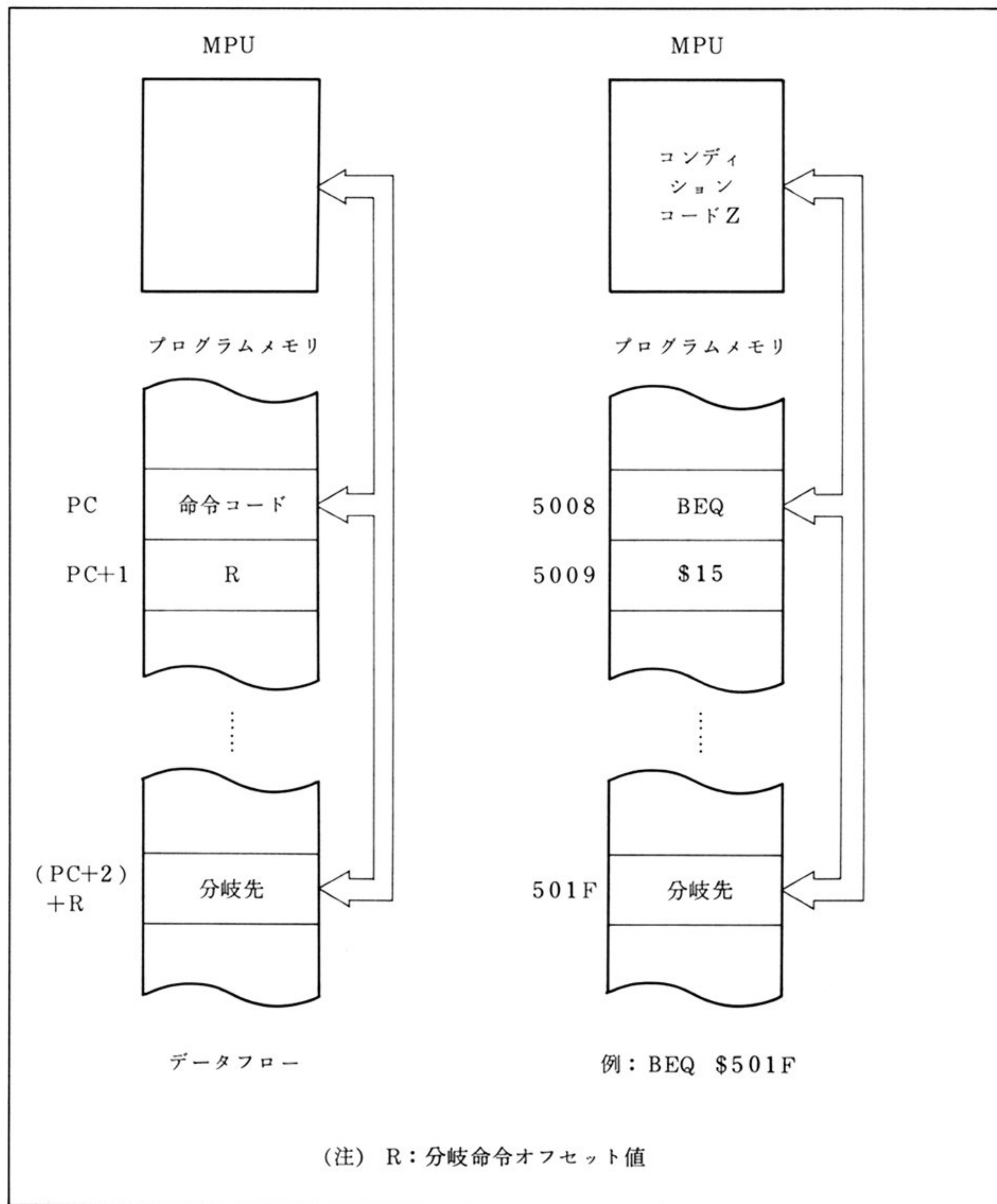



図 4.11 相対番地指定(1)の処理形態

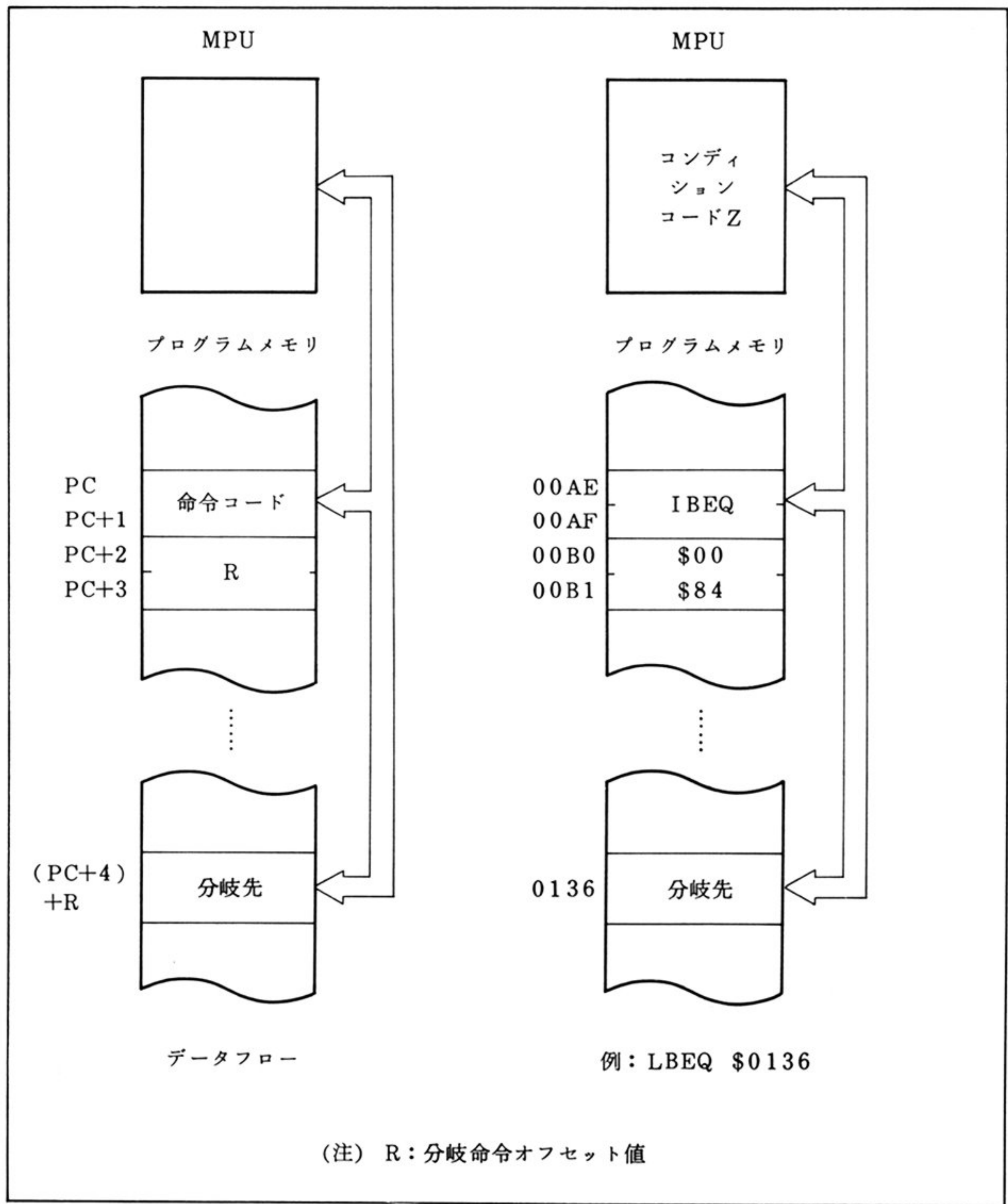



図 4.12 相対番地指定(2)の処理形態

```
(ロケーション(10進))   (文)
      0000    LAB1    EQU    *
                       :
      0130    LAB2    EQU    *
                       :
                      BLE    LAB1      (誤った例)
                       :
      0200            BNE    LAB2      (正しい例)
      0202            LBRA   LAB3      (正しい例)
                       :
      0332    LAB3    EQU    *
                       :
                      END
```

図 4.13　相対番地指定の記述例

4.2.6 直接番地指定と拡張番地指定

直接番地指定及び拡張番地指定は，これらの番地指定が許されている命令のオペランド欄に式だけ(#や，レジスタは記述不可)を記述することによって指定され，翻訳の結果，オペランド欄の式の値が番地として番地部に設定されます．

直接番地指定の番地部は1バイト(8ビット)の符号なし2進数として表わされ，拡張番地指定の番地部は2バイト(16ビット)の符号なし2進数として表わされます．

オペランドの式が次の場合のとき直接番地指定が用いられます．

(1) オペランドの式の最上位バイトの値が現在の直接ページ擬似レジスタの値と等しい式を参照しているとき．但し，式が前方参照記号を含んでいるとフェーズエラーを回避するために，拡張番地指定で翻訳されます．

(2) オペランドの先頭に"＜"を付けたとき．但し，オペランドの式の最上位バイトの値が現在の直接ページ擬似レジスタの値と一致していなければなりません．

前記以外の場合は，拡張番地指定が用いられます．

また，オペランドの先頭に“ > ”を付けることにより強制的に拡張番地指定にすることができます．

特に，拡張番地指定で間接形式を用いることができます．

一般形　　　　〔式〕

で表わされ，式の値を含んだ2バイトのオフセットとともに，1バイトのポストバイトを生成します．

直接番地指定及び拡張番地指定の処理形態を図4.14と図4.15に，記述例を図4.16に示します．

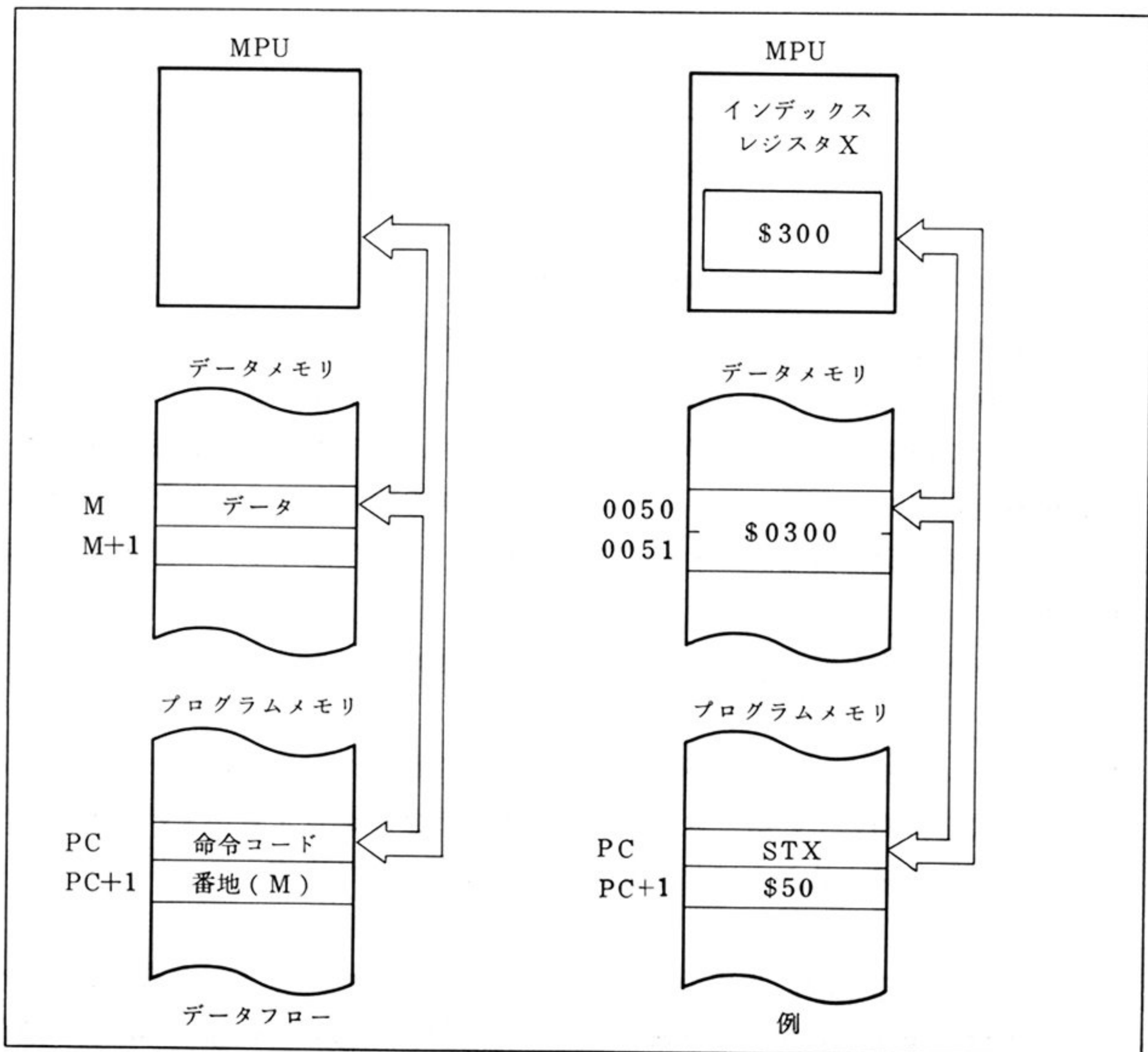



図4.14　直接番地指定の処理形態

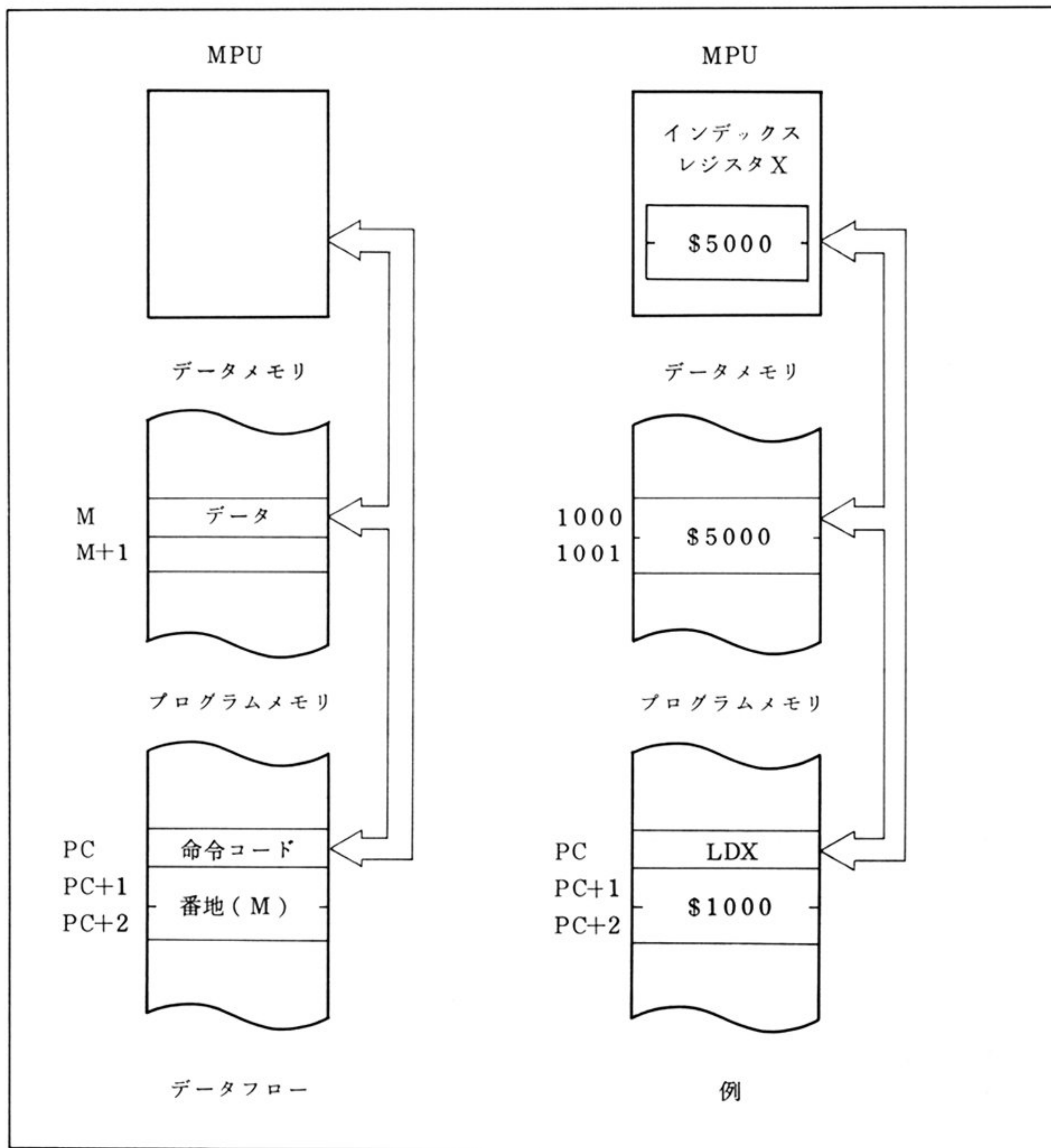



図4.15　拡張番地指定の処理形態

```
（ロケーション(16進））
                              SETDP    $00
        0080      B1          FDB      $100
                                ⋮
        0100      D1          RMB      2
                                ⋮
                              LDX      255     （直接番地指定）
                              SETDP    $10
                              STX      D1      （直接番地指定）
                                ⋮
                              LDX      B1      （拡張番地指定）
                              SETDP    $04
                              LDX      B2      （B2が前方参照
                                ⋮                記号であるため,
                                                 拡張番地指定）
        0400      B2          FDB      $200
```

図 4.16　直接番地及び拡張番地指定の記述例

4.2.7 レジスタ番地指定

レジスタ番地指定は，オペランド欄にMPU内のレジスタ（A，B，CC，D，DP，PC，S，U，X，Y）を記述することによって指定されます．

レジスタ番地指定による基本命令はTFR命令，EXG命令，PSH命令，PUL命令の4つがあります．

次にこれらの命令による形式について説明します．

(1) TFR／EXG命令の場合の形式

TFR／EXG命令の場合は，オペランド欄にレジスタ並びとして，A，B，CC，D，DP，PC，S，U，X，Yレジスタのうちどれか二つだけを記述する形式です．

TFR命令では，指定する二つのレジスタの大きさは同じ大きさにするか，転送元を16ビットのレジスタ（D，PC，S，U，X，Yレジスタ），転送先を8ビットのレジスタ（A，B，CC，DPレジスタ）にしなければなりません．

EXG命令では，指定する二つのレジスタの大きさは同じでなければなりません．

TFR / EXG 命令両方ともオペランド欄に記述したレジスタによって，命令コードに続く直接数値（ポストバイト）が決定されます．

ポストバイトについては，「付録2　ポストバイトの形式」を参照してください．

(2)　PSH / PUL命令の場合の形式

PSH/ PUL 命令の場合は，オペランド欄にレジスタ並びとして，A , B , CC , D , DP , PC , S , U , X , Y レジスタのうちどれでも指定できます．また，この形式では次のことに注意すればレジスタを複数個指定できます．

U レジスタは，PSHU 命令及び PULU 命令と一緒に指定できません．同様に，S レジスタは PSHS 命令及び PULS 命令と一緒に指定できません．

TFR / EXG 命令同様，オペランド欄に記述したレジスタ並びによって，ポストバイトが生成されます．

次に，レジスタ番地指定の処理形態を図4.17と図4.18に，記述例を図4.19に示します．

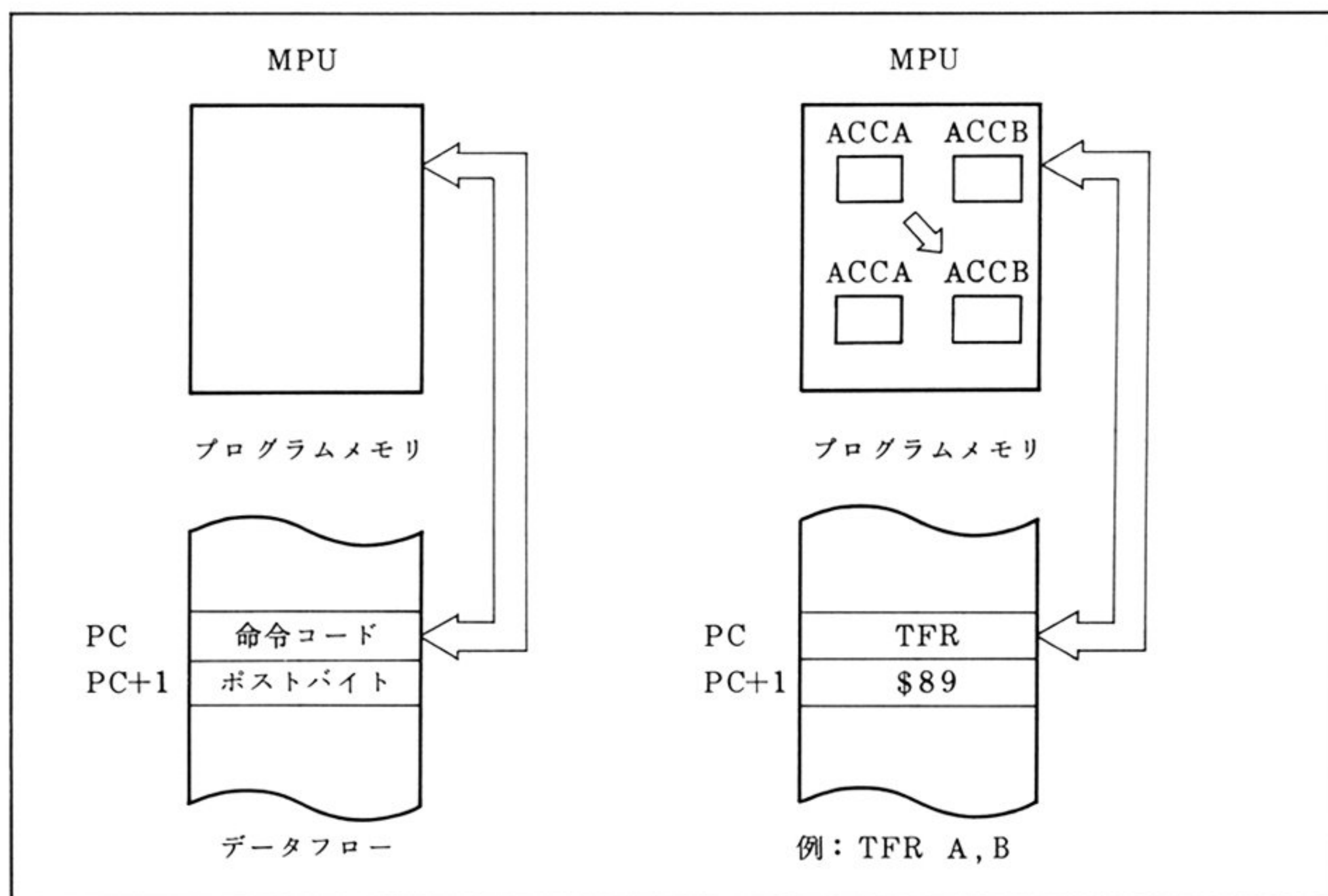



図4.17　レジスタ番地指定(1)の処理形態

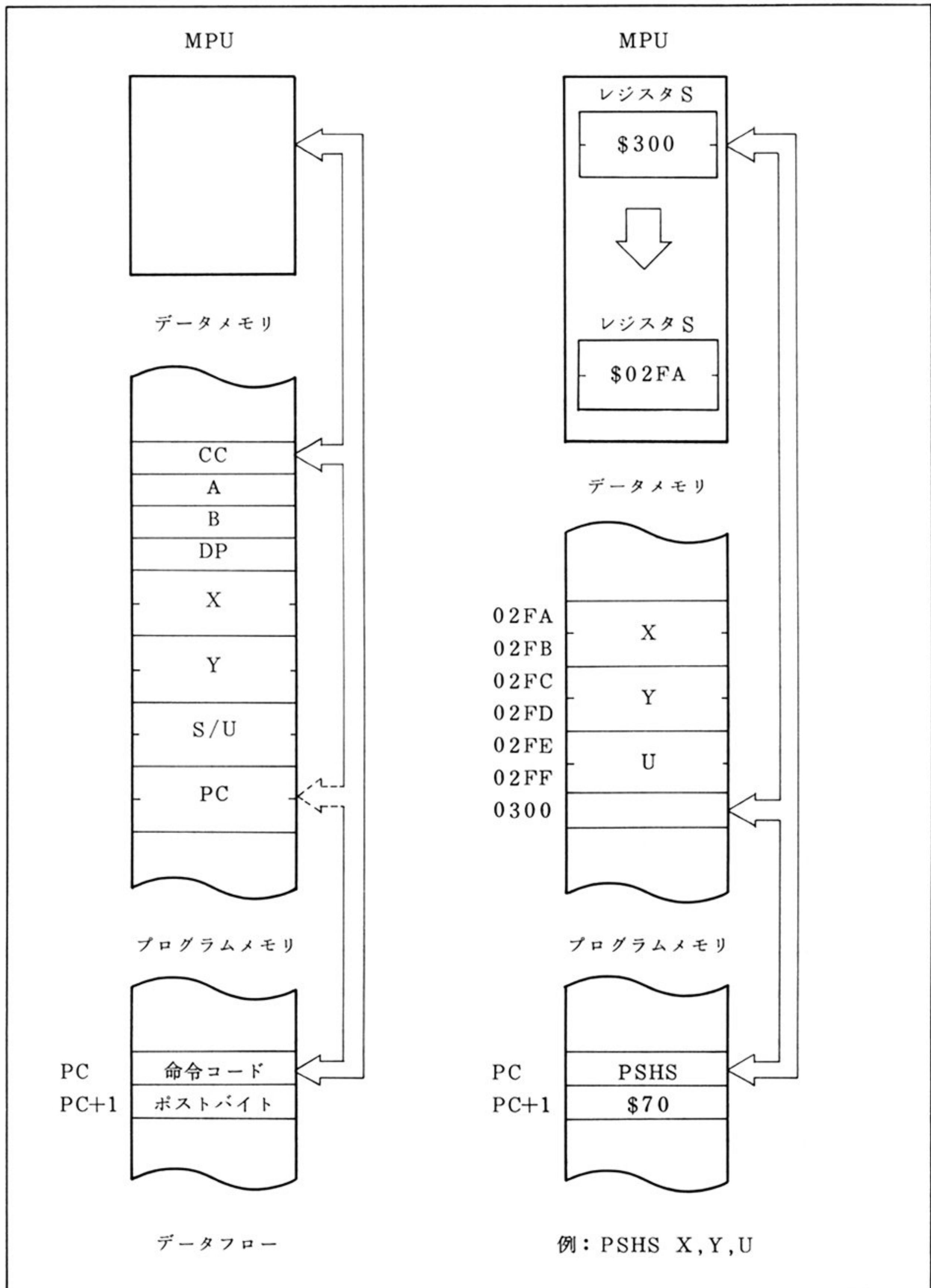



図 4.18　レジスタ番地指定(2)の処理形態

```
〔正しい例〕
    EXG     A,B
    EXG     X,Y
    TFR     X,A          (警告が起るが正しい)
    PSHS    X,Y,U
    PULS    A,B,X,Y
〔誤った例〕
    EXG     X,A          (二つのレジスタの大きさが違う)
    TFR     A,X          (8ビットレジスタから16ビットレジスタ
                          への転送はできない)
    TFR     A,B,X        (レジスタの数が多すぎる)
    PSHS    X,Y,S        (PSHS命令でSレジスタは指定できない)
```

図 4.19　レジスタ番地指定の記述例

4.3 命　令　形　式

機械命令には，番地指定を行なうためのオペランド番地指定形式が一つに固定されている命令と，オペランドの記述形式によって二つ以上の中から選択できる命令とがあります．後者に属する命令は，翻訳されて作り出される機械命令の形式が，番地指定形式によって異なります．

図 4.20 に機械命令文と，それが翻訳されて作り出される機械命令の例を示します．

図 4.20　機械命令文とそれに対応する機械命令の例

第5章　制御命令

制御命令には，プログラムの翻訳を制御する命令（NAM，END），番地を指定する命令（ORG，SETDP）及び目的プログラムの形式，リストの制御を行なう命令（OPT，TTL，PAGE（PAG），SPC）があります．

5.1　NAM命令

NAM命令は，一般にソースプログラムの始まりを示すとともに，それに名前を与える命令です．

〔形　式〕

	NAM	プログラム名

〔規　則〕

(1)　オペランド欄に1～6文字のプログラム名を書くことができます．このプログラム名は，ソースプログラム及び目的プログラムリストの各ページの先頭に印刷されます．

(2)　NAM命令は，通常ソースプログラムの始めに記述しますが，途中に記述することもでき，省略もできます．なお省略したときはプログラム名は空白となります．

(3)　ラベル欄は記入できません．

5.2　END命令

END命令は，アセンブラにソースプログラムの終わりを指示します．

〔形　式〕

	NAM	式

〔規　則〕

(1) END命令は，オペランド欄に式を書くことができます．この式の値は，通常プログラムの実行開始番地を表します．
(2) END命令以後に記述されている文は，翻訳されません．
(3) ラベル欄は記入できません．

5.3 ORG命令

ORG命令は，ロケーションカウンタを強制的にオペランド欄に書かれた式の値に変更する命令です．

〔形　式〕

	ORG	式

〔規　則〕

(1) オペランド欄の式で使用する記号は，このORG命令が出現する以前に，既に定義されていなければなりません．
(2) ロケーションカウンタの値がオーバラップするようなORG命令を使用してはなりません．
(3) ORG命令によりロケーションがスキップされた場合アセンブラ終了後，その目的プログラムをF-BASICのLOADMコマンドを使用してメモリにロードすると，スキップされたロケーションに対応するメモリの値は不定となります．

次にORG命令の使用例を図5.1に示します．

図 5.1　ORG 命令の使用例

5.4 SETDP命令

SETDP 命令は，翻訳時の直接ページ擬似レジスタに値を割り当てるために使用します．式の最下位バイトの値が擬似レジスタに割り当てられ，特定のメモリ参照が直接番地指定を用いることができるかどうかを決定するのに用いられます．

翻訳の始めに，直接ページ擬似レジスタにはゼロが割り当てられます．

〔形　式〕

	SETDP	式

〔規　則〕

(1)　SETDP 命令は，翻訳中に何回でも使用することができ，そのたびに直接ページ擬似レジスタを更新します．

(2)　式は，前方参照記号あるいは未定義記号を含むことはできません．

(3)　式の最上位バイトの値がゼロでないと警告が起きます．しかし，それでも直接ページジレジスタには，式の最下位バイトの値が割り当てられます．

(4) SETDP命令は，実行時の直接ページレジスタに影響を及ぼさないように注意し，翻訳時と実行時の値が矛盾しないよう注意を要します．

例えば，　　SETDP　$20

は，直接ページ擬似レジスタを$20に設定し，その結果絶対番地$2000～$20FFの番地参照は，直接番地形式に翻訳されます．

5.5 OPT命令

OPT命令は，プログラマがアセンブラの出力形式を制御するための命令です．オプションは，オペランド欄にコンマで区切って書かれ，各オプションの記述順序は任意ですが少なくとも一つは書かなくてはなりません．ここでは，出力リストの制御のためのオプションと翻訳制御のためのオプションを併わせて説明します．

〔形　式〕

```
    OPT  [{OBJ   }] [,{MEM    }]
          {NOOBJ }    {NOMEM  }

         [PAGE=n  ] [,LLEN=n  ]

         [{LIST  }] [,{GEN    }]
          {NOLIST}    {NOGEN  }

         [{SYM   }] [,{W      }]
          {NOSYM }    {NOW    }
```

(1) 下線のオプションは，これらのオプションの指定がなかったとき，アセンブラが採用する省略時のオプションを示します．

(2) オプションの上のピリオドは，そのオプションの短縮形を示します．すなわち，オプションを指定するとき，すべての文字を指定するかわりに，ピリオドの付いた文字のみ短縮形として指定することもできます．

（例）　NOO　　（NOOBJと同じ）

　　　　P=60　　（PAGE=60と同じ）

〔規　則〕

(1) $\left\{\frac{\text{OBJ}}{\text{NOOBJ}}\right\}$

このオプションは，翻訳結果の目的プログラムをファイルへ出力するかどうかを指示します．OBJ 指定をすると目的プログラムはファイルへ出力され，NOOBJ 指定をすると目的プログラムはファイルへ出力されません．

(2) $\left\{\frac{\text{MEM}}{\text{NOMEM}}\right\}$

MEM指定は，目的プログラムをメモリに直接出力することをアセンブラに指定します．

目的プログラムの出力される領域は，アセンブラが使用しているメモリ領域と連続してはなりません．

NOMEMオプションは，目的プログラムをメモリに直接出力しません．

(3) PAGE＝n　（10≦n≦255）

PAGE＝n（nは10進数で10≦n≦255）指定は，ページ形式で印刷を行ない，1ページの行数をnで指定された直にすることをアセンブラに指示します．このオプションの指定が省略されると1ページに58行印刷されます．

(4) LLEN＝n　（50≦n≦136）

アセンブルリストの印刷を行なうとき，1行の長さをnで指定します．このオプションの指定が省略されると1行は79けたに設定されます．

(5) $\left\{\frac{\text{LIST}}{\text{NOLIST}}\right\}$

このオプションは，ソースプログラム及び目的プログラムリスト（「8.5.1 ソースプログラム及び目的プログラムリスト」参照）を出力するかどうかを指示するものです．

LISTが指定されるとソースプログラム及び目的プログラムリストは出力され，NOLISTが指定されると出力されません．

なお，NOLISTが指定されると，GENオプションは指定されていても無効

となり，LISTが指定されているときのみ有効となります.

(6) $\left\{\begin{array}{l}\underline{\text{GEN}}\\\text{NOGEN}\end{array}\right\}$

ソースプログラム及び目的プログラムリストを印刷するとき，FCC,FCB，FDB命令の展開形のすべてを印刷するかどうかを指示するものです．GENが指定されたときは展開形のすべてが印刷され，NOGENが指定されたときは先頭の1バイト又は2バイト（FDB命令の場合）のみ印刷されます.

(7) $\left\{\begin{array}{l}\underline{\text{SYM}}\\\text{NOSYM}\end{array}\right\}$

アセンブラが作成した記号テーブルリストを出力するかどうかを指示するものです．SYMが指定されたときは記号テーブルリストは出力され，NOSYMが指定されたときは出力されません．記号テーブルリストの詳細については「8.5.2　記号テーブルリスト」を参照してください.

(8) $\left\{\begin{array}{l}\underline{\text{W}}\\\text{NOW}\end{array}\right\}$

ソースプログラム及び目的プログラムリストを印刷するとき，警告メッセージを印刷するかどうかを指示するものです．Wが指定されたとき警告メッセージが印刷され，NOWが指定されたときは印刷されません.

5.6 TTL命令

TTL命令は，オペランド欄に記入された文字列をタイトルとして，ソースプログラム及び目的プログラムリストの各ページの見出しに印刷することを指示します.
このタイトルは，次にTTL命令により文字列が指示されるまで有効です.

〔形　式〕

	TTL	文字列

〔規　則〕

(1) 文字列は最大45文字です.

(2) ラベル欄は記入できません.

5.7 PAGE(PAG)命令

PAGE（PAG）命令は，ソースプログラム及び目的プログラムリストを次のページの先頭に進める命令です.

〔形　式〕

〔規　則〕

(1) この命令は，ラベル欄，オペランド欄ともに空白です.

(2) この命令自身は，リスト上に印刷されません.

5.8 SPC命令

SPC命令は，ソースプログラム及び目的プログラムリストを印刷するとき，空白行を作ることを指示する命令です．オペランド欄に記述された式の値だけ空白行が作成されますが，SPC命令によってページ境界を越える場合はPAGE命令と同じ働きをします.

〔形　式〕

	SPC	式

〔規　則〕

(1) 式にはまだ定義されていない記号が含まれていてはなりません.

(2) 式の値は1以上255以下でなければなりません.

(3) この命令自身は，リスト上に印刷されません.

(4) ラベル欄は記入できません.

第6章　記号定義命令

アセンブラ言語は，記号化された言語です．記号は，種々の目的に使用されます．たとえば，命令欄に記述した記号は，機械命令あるいはアセンブラ命令を示します．

記号は，一般的にラベル欄で定義されますが，定義された記号には値が割り当てられます．このようにして割り当てられた値は，オペランド欄で参照することが可能となります．

この章では，このような記号の使用方法を，更に促進するための EQU 命令及び REG 命令について記述されています．

6.1 EQU命令

EQU 命令は，ラベル欄に書かれた記号にオペランド欄の式の値を割り当てます．EQU 命令は，プログラムのロケーションカウンタ以外の値を得る唯一の方法です．

〔形　式〕

記　号	EQU	式

〔記　号〕

(1) 式の中で項として使用した記号は，この命令より以前の文で定義されていなければなりません．

(2) 一度定義された記号は，再定義してはいけません．

次に EQU 命令の使用例を図 6. 1 に示します．

```
         NAM   EQU
         OPT   SYM
L1       RMB   100
L2       EQU   L1+$20
L3       EQU   L2+$10
V1       EQU   $0D
          ⋮
         END
```

図 6.1　EQU 命令の使用例

図6.1においては記号L1のロケーションカウンタ値は0です．したがってL2には$(20)_{16}$，L3には$(30)_{16}$の値がそれぞれ割り当てられます．またV1には$(0D)_{16}$の値が割り当てられます．

6.2　REG命令

REG命令は，プログラムカウンタ以外のレジスタ並びと関連づけられた値を記号に割り当てる命令です．

〔形　式〕

記　号	REG	＜レジスタ並び＞

ここで，＜レジスタ並び＞はR_1〔，R_2，…Rn〕の形式で，Ri（i＝1～n）は，A，B，CC，D，DP，PC，S，U，X，Yレジスタのうちの一つを表す．

〔規　則〕

(1)　ラベル欄の記号はプログラム中のどんな場所においても再定義されてはなりません．

(2)　レジスタ並び中に同じレジスタを2回以上記述すると警告が起ります．
レジスタDは，レジスタA及びレジスタBと同じです．

(3)　記号は，どんな式の中で用いてもよいが，その値はPSH命令，PUL命令と共に用いられたときのみ意味を持ちます．

このPSH/PUL命令には次の二つの形式があります.

$$\left\{\begin{matrix}\text{PSHU}\\\text{PULU}\\\text{PSHS}\\\text{PULS}\end{matrix}\right\}$$ ＜レジスタ並び＞　　又は　$$\left\{\begin{matrix}\text{PSHU}\\\text{PULU}\\\text{PSHS}\\\text{PULS}\end{matrix}\right\}$$ #＜レジスタ式＞

ここで，＜レジスタ式＞はLAB1〔!+LAB!+…!+LABn〕の形式で，LABi（i＝1～n）はREG命令で定義された記号でなければなりません.

「4.2.7　レジスタ番地指定」で述べたように，UレジスタはPSHU命令及びPULU命令とは一緒に用いることはできないので，PSHU命令及びPULU命令のとき＜レジスタ式＞で用いる記号に，Uレジスタを含む記号を用いることはできません.

また，PSHS命令及びPULS命令とSレジスタの関係も同じことが言えます.

次に，REG命令の使用例を図6.2に示します.

```
〔正しい例〕
    ALLREG  REG   A,B,CC,DP,X,Y,PC,U
    REGXY   REG   X,Y
    REGAB   REG   A,B
            PSHS  #ALLREG
            PULU  #REGXY!+REGAB
〔誤った例〕
   REGUS    REG   U,S     （UレジスタとSレジスタの両方は指定できない）
   REGU     REG   U
            RSHU  #REGU （ユーザスタック上にUレジスタはプッシュできない）
   REGLST   REG   A,B,D （レジスタ名が重複しているDはA，Bと同じである）
            PSHS  #REGU!+REGU （レジスタ名が重複している）
```

図6.2　REG命令の使用例

第7章　データと領域の定義

データ定義命令とは，記憶域にある定数あるいは領域を確保するアセンブラ命令の総称です．データ定義命令には，定数定義命令（FCC命令，FCB命令，FDB命令）と領域定義命令（BSZ命令，RMB命令）の2種類があります．これらの命令のラベル欄には記号を定義することができます．プログラムは，この記号を機械命令やアセンブリ命令のオペランド欄に記述することにより参照することができます．データ定義命令で定義した定数や領域は，実行時に機械命令で参照及び変更が行なわれます．

7.1 FCC命令

FCC命令は，文字列をASCIIコードに変換し連続した記憶域に格納するものです．

〔形　式〕

〔記号〕	FCC	{ n文字列 d文字列d }

(1) nは文字列の文字数を表わし，1～255の符号なし10進数です．

(2) dは区切り記号であり，文字列の中に含まれない任意の文字を記述します．

〔規　則〕

(1) 文字列に含まれる文字は，ASCII 16進コードの20（スペース）から，5F（￣）に対応するどんな文字でも使用できます．

(2) nが文字列の文字より大きいときは，文字列の後に空白が補なわれます．

(3) 文字列の表現方法は，前述の2通りの方法がありますが，どちらにも解釈できる場合は上の表現として翻訳されます．

次にFCC命令の記述列を図7.1に示します．

```
            :
D1    FCC   /**FCC1**/
      FCC   $**FCC2**$
      FCC   !**FCC3*!!                 ①
*
D2    FCC   4,ABCD
      FCC   4,E
      FCC   8,FM-8 ASSMBLER            ②
```

図7.1　FCC命令の記述例

図7.1の①のように，文字列中に区切り記号が現われた場合はその区切り記号以後の文字（！）は注釈とみなされます．

また，②のように，指定文字数よりも文字列の文字数が大きい場合には，指定文字数以後の文字は注釈と見なされます．

7.2 FCB命令

FCB命令は，一つあるいはいくつかの1バイト（8ビット）の定数を，記憶域に連続して設定する命令です．定数は式で表し，オペランド欄にコンマ（，）で区切って記述します．

〔形　式〕

〔記号〕	FCB	{{式 空白}〔式， ， 式〕〔…〕〔式〕}

〔規　則〕

(1) 式の値は，8ビットの符号付き2進数で表現され，その値が8ビットより大きい場合は，下位8ビットが有効となります．

(2) 式の値が0のときは，式そのものを省略することができます．ただし，オペランドすべてを省略することはできません．

次にFCB命令の記述例を図7.2に示します．

(ロケーション(16進))	(データ)		(文)	
			⋮	
			OPT	SYM
6100			ORG	$6100
6100	20		FCB	$20
6101	30		FCB	$30,,
6102	00			
6103	00			
6104	00		FCB	,$40
6105	40			
	10	Z	EQU	$10
6106	0A		FCB	10,Z+$20
6107	30		⋮	

図7.2 FCB命令の記述例

7.3 FDB命令

FDB命令は，一つあるいはいくつかの2バイト(16ビット)の定数を記憶域に連続して設定する命令です．定数は式で表し，オペランド欄にコンマ(，)で区切って記述します．

〔形 式〕

〔記号〕	FDB	{{式，/空白，}〔式，/，/式〕〔…〕〔式〕}

〔規 則〕

(1) オペランド欄には，一般の式を記述できます．

(2) 式の値は，16ビットの符号なし2進数として表現されます．

(3) 式の値が0のときは，式そのものを省略することができます．ただし，オペランドすべてを省略することはできません．

次にFDB命令の記述例を図7.3に示します．

(ロケーション(16進))	(データ)		(文)	
			⋮	
			OPT	SYM
6100			ORG	$6100
6100	000F	Z	FDB	$F,
6102	0000			
6104	6110		FDB	Z+$10
6106	6101		FDB	Z+1
6108	2345		FDB	$12345
			⋮	

図7.3　FDB命令の記述例

7.4　BSZ命令

BSZ命令は，初期値がゼロ(00)₁₆の連続した記憶域を割り付けます.割り付けられるバイト数は，オペランド欄の式で与えられます.

〔形　式〕

〔記号〕	BSZ	式

〔規　則〕

(1)　式には，まだ定義されていない記号を含んでいてはなりません.

(2)　式の値が0のときは，記憶域は割り付けられません.

次にBSZ命令の記述例を図7.4に示します.

(ロケーション(16進))	(データ)		(命令文)	
			⋮	
			OPT	SYM
6100			ORG	$6100
6100	0050		BSZ	$50
6150	000A		BSZ	10
	000A	W	EQU	10
	000A	Z	EQU	10
615A	0064		BSZ	W*Z
61BE			⋮	

図7.4　BSZ命令の記述例

7.5 RMB命令

RMB命令は，オペランドで示される式だけ，記憶域をバイト単位で確保する命令です．また確保された領域は初期化されません．

〔形　式〕

〔記号〕	RMB	式

〔規　則〕

(1) 式には，まだ定義されていない記号を含んでいてはなりません．

(2) 式の値が0のときは，記憶域は確保されません．

次にRMB命令の記述例を図7.5に示します．

(ロケーション(16進))	(データ)		(文)	
			⋮	
			OPT	SYM
6200			ORG	$6200
6200	0050		RMB	$50
	0014	W	EQU	20
	000A	Z	EQU	10
6250	00C8		RMB	W*Z
6318	0004		RMB	4
			⋮	

図7.5　RMB命令の記述例

第8章　使用手引

ソースプログラムを，目的プログラムに変換する一連の手順を翻訳するといいます．本章では，翻訳する場合に必要な入出力機器，翻訳の結果として作り出される出力リストなどについて説明します．

8.1 機　器　構　成

翻訳を行なうのに必要な機器の構成を図8.1に示します．なお，ソースプログラムの翻訳を行なうのに当り，ソースプログラムは必ずディスク上にあらかじめ登録されていなければなりません．

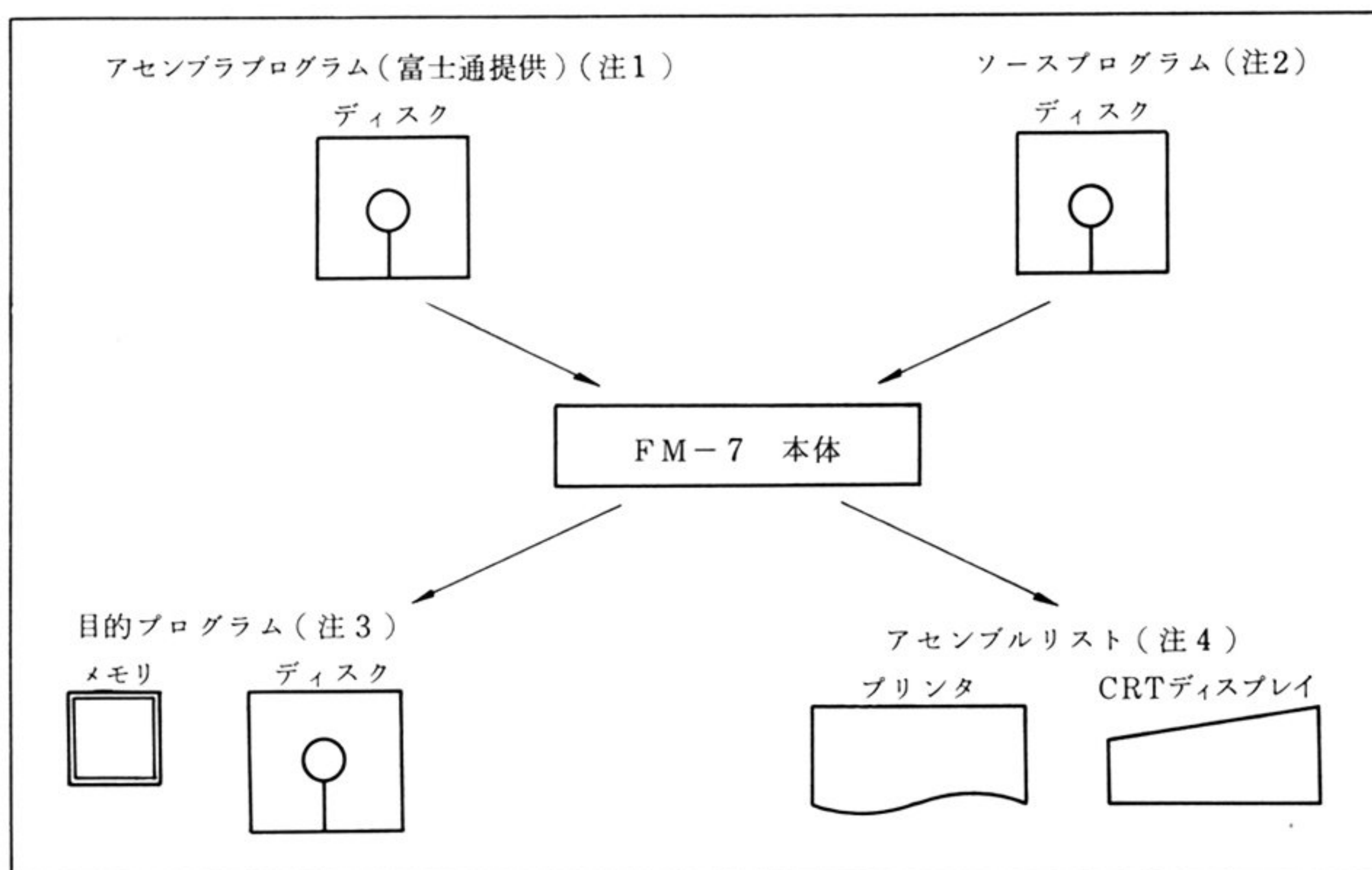



図8.1　翻訳を行うのに必要な入出力機器構成

（注1）　富士通が提供するアセンブラプログラムが格納されているディスクを示します．
アセンブラプログラムは，ディスク上では二つのファイルから構成されています．一番目のファイル名は“ASM09”というファイル名を持ったBASIC言語で記述されたプログラムフ

ァイルです.

二番目のファイル名は"ASM09EB"というファイル名を持ったMBL6809アセンブラ言語を使用して作られた機械語ファイルです.

ユーザは，アセンブラプログラムを購入された場合は，念のためそのコピーを作成しておくことを推奨します.

(注2) ソースプログラムが登録されているディスクを示します.

なお，ソースプログラムファイルの作成は，通常F－BASICのエディット機能を使用してソースプログラムをメモリ上に作成し，それをF－BASIC の SAVE コマンドを用いて，ASCII形式でファイルに格納することにより行なわれます.（「8.3 ソースプログラムファイルの作成」を参照）.

(注3) 目的プログラム（MBL6809の機械語プログラム）が出力されるディスクを示します．メモリに直接目的プログラムを出力するためには，アセンブラのOPT命令でMEMオプションを指定します（「5.5 OPT命令」を参照）.

(注4) アセンブルリストの出力媒体（プリンタ又はCRT ディスプレイ）を示します.

8.2 翻訳の手順

ソースプログラムの翻訳は，F－BASIC の管理下で実行されます．ディスクモードでF－BASIC を起動させる際は，ディスクドライブ数及びディスクファイル数は4以下の数を指定しなければなりません．これ以上の指定をすると，アセンブラをRUNするのに充分なメモリ領域が確保できなくなり，"Out of Memory"のエラーになります.

ユーザは翻訳を開始する前に，F－BASIC のDATE$及びTIME$関数を用いて日付け及び時刻を設定しておけば，翻訳時のアセンブルリスト上に日付け及びアセンブル開始時刻を表示することができます.

8.2.1 翻訳の操作手順

翻訳を行なうためには，アセンブラプログラム（富士通提供）のメモリへのロード，そして翻訳の制御データとして翻訳すべきソースプログラムファイル，アセンブルリスト装置及び目的プログラムファイルの指定を行なわなければなりません.

以下に順を追って翻訳の手順を説明します．なお，説明の中で下線の部分はユーザがキーボードから入力しなければならないことを示しています.

(1) 富士通提供のアセンブラプログラムが格納されている媒体をカセット装置，バブルユニット #0 又はディスクドライブ #0 にセットします.

(2) 続いて次のF－BASIC コマンドを入力します.

```
RUN"0：ASM09"
```

このRUNコマンドによりアセンブラが入力媒体よりメモリへロードされ翻訳が開始されます.

(3) 最初にアセンブラの開始メッセージが次のように表示されます.

```
FM－7 MBL6809 ASSEMBLER
V1．1 BEGIN.
```

(4) 続いて，翻訳すべきソースプログラムのファイルディスクリプタの指定がアセンブラにより要求されますので，ユーザはファイルディスクリプタの入力を行ないます.

```
SURCE FILE DESCRIPTOR ?＝"n：ファイル名"
```

- "n：ファイル名"………ソースプログラムをディスクドライブユニットn（n＝0～7）にセットされているディスク上のファイル名で示されるファイルから入力することを示します.

ソースファイルディスクリプタの指定が正しく行なわれ，ソースプログラムファイルがオープンされると，次のメッセージが表示されます.

```
SOURCE FILE WAS OPENED.
```

(5) 続いて，アセンブルリストをどの装置に出力するか，出力装置の指定が要求されますので，ユーザは出力装置の指定を行ないます.

```
                     ⎧ "SCRN:"
LIST DEVICE ?＝      ⎨ "LPT0:"
                     ⎩ 
```

- "SCRN:"………アセンブルリストをCRTディスプレイに出力することを示します.
- "LPT0:"………アセンブルリストをプリンタに出力することを示します.

・⏎ ………何にも指定しない場合には，アセンブルリストを出力しないことを示します．なお，これはソースプログラム中に記述するOPT命令のLIST/SYMオプションより優先します．即ち，ソースプログラム中のOPT命令でLIST/SYMオプションが指定してあってもLIST DEVICEを指示しなければアセンブルリストは出力されません．

LIST DEVICEとして"LPT0："（プリンタ装置）を指定した場合，プリンタ装置が正しくオープンされると，次のメッセージが表示されます．

```
LINE PRINTER WAS OPEND.
```

(6) 続いて，出力すべき目的プログラムのファイルディスクリプタの指定が要求されますので，ユーザはファイルディスクリプタの入力を行ないます．

```
OBJFCT FILE DESCRIPTOR ?= { "n：ファイル名"⏎ }
                          { ⏎               }
```

・"n：ファイル名"………目的プログラムをディスクドライブユニットn（n＝0～7）にセットされているディスク上のファイル名で指示されるファイルへ出力することを示します．

・⏎ ………何にも指定しない場合は，目的プログラムをファイルへ出力しないことを示します．なお，これはソースプログラム中に記述するOPT命令のOBJオプションより優先します．

目的プログラムのファイルディスクリプタが正しく指定され，目的プログラムファイルがオープンされると次のメッセージが表示されます．

```
OBJECT FILE WAS OPENED.
```

(7) 以上，三つの制御データの入力が終わるとアセンブラのパス1の処理が開始されます．パス1の処理が終了するとパス2の処理が自動的に開始されます．

(8) パス2の処理が終了すると最初に次のメッセージが表示されます．

```
TOTAL ERRORS eeeee－－mmmmm
TOTAL WARNINGS wwwww－－nnnnn
 eeeee：エラーの個数
 wwwww：警告の個数
 mmmmm：エラーが最後に起きた文の行番号
 nnnnn：警告が最後に起きた文の行番号
```

(9) 続いて，次のメッセージが表示されます．

```
PROGRAM START ADDR = ssss
PROGRAM END   ADDR = eeee
PROGRAM ENTRY ADDR = pppp
```

ssss,eeee（16進数）は，翻訳された目的プログラムの先頭番地，終了番地を示します．

pppp（16進数）は，ソースプログラム中のEND命令のオペランドの値でプログラムの入口番地を示します．オペランドが記述されていない場合は，4個のアスタリスク（＊＊＊＊）が表示されます．

(10) 最後に

```
ASSEMBLER END.
```

を表示して，翻訳を終了します．

なお，翻訳終了時点において，F－BASICが使用できるメモリの上限値は17FF（16進数）になっています．

また，アセンブラのOPT命令でMEMオプションが指定されているならば，目的プログラムはメモリ中に存在しています．メモリ中に存在している目的プログラムは，F－BASICのSAVEMコマンドを使用して，カセットファイル等に記録することもできます．もし，引き続き他のソースプログラムの翻訳を続行したい場合は，

コマンドにより(3)から再び翻訳を実行することができます．

8.2.2 操作手順中に出力されるエラーメッセージ

「8.2.1 翻訳の操作手順」において入力された制御データに誤りがあると，アセンブラは以下に示すエラーメッセージを表示して，制御データの再入力を要求してきます．ユーザはエラーの原因を取り除いて正しい制御データを入力すれば，操作は続行されます．

(1) ****ERROR Bad File Descriptor

ファイルディスクリプタの形式指定に誤りがあることを示しています．

(2) ****ERROR File Not Found

指定したソースプログラムファイルが存在しなかったことを示しています．

(3) ****ERROR File Already Exists

指定した目的プログラムファイルと同じ名前のものが，既に存在していたことを示しています．

(4) ****ERROR Drive Not Ready

指定されたドライブ番号の装置がReady 状態でなかったことを示しています．

(5) ****ERROR Inavalid List Device

アセンブルリストの出力装置の指定に誤りがあることを示しています．

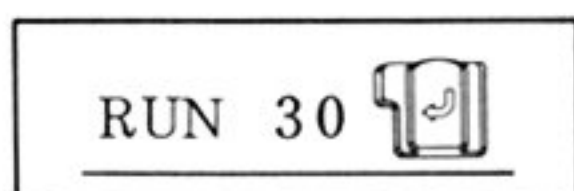
(6) ****ERROR Device In Use

使用中のデバイスに対して再度オープンしようとしたことを示しています．

(7) ****ERROR Not Ascii File

指定したソースプログラムファイルがバイナリファイルであることを示しています．

なお，上記のエラーメッセージ以外にF－BASICが入出力に関するエラーメッセージを表示して翻訳の操作が異常終了してしまう場合がありますが，その場合には，

RUN 30 ⏎

コマンドにより「8.2.1　翻訳の操作手順」の(3)の所に戻って翻訳を再開することができます.

次に，翻訳の操作手順例を図8.2に示します.

8.3 ソースプログラムファイルの作成

ソースプログラムは，F－BASICのプログラムを作るときの要領で初めにメモリ中へ作成します．次にF－BASICのLISTコマンドを用いて，各ステートメントの目視チェックを行ない，そこで誤りが見つかれば，EDITコマンド等を用いてステートメントを修正します．最後にSAVEコマンドを用いて，ASCII形式でソースプログラムをファイル（ディスク）に格納します.

なお，アセンブラでは文字セットとして英小文字も認めていますが，注釈行，注釈欄，FCC命令のオペランド及び文字自己規定項を除く箇所で用いられた英小文字は，英大文字と解釈されます.

次に，ソースプログラムファイルの作成例を図8.3に示します.

Ready DATE$="83/06/01"	日付けの設定
Ready TIME$="10:30:00"	時刻の設定
Ready RUN"ASM09"	ディスクに格納されているアセンブラプログラムをメモリへロードし，翻訳を開始させる.
FM-7 MBL6809 ASSEMBLER V1. 1 BEGIN.	翻訳開始メッセージ
SOURCE FILE DESCRIPTOR ? = "TEST" SOURCE FILE WAS OPENED. LIST DEVICE ? = "SCRN:" OBJECT FILE DESCRIPTOR ? ="TESTEB" OBJECT FILE WAS OPENED.	翻訳制御データの入力及び出力メッセージ • ソースプログラムは，ディスクに格納されている"TEST"というファイル名のプログラムである. • リスト出力は，CRTディスプレイである. • 目的プログラムを"TESTEB"というファイル名に出力する.
PAGE 001 (830601,103312) TEST 00010 NAM TEST 00020 6000 ORG $6000 00030 6000 START EQU * ⋮ ⋮ ⋮ ⋮ 00730 6073 39 RTS 00740 6000 END START TOTAL ERRORS 00000--00000 TOTAL WARNINGS 00000--00000 PROGRAM BEGIN ADDR=6000 PROGRAM END ADDR=6073 PROGRAM ENTRY ADDR=6000	アセンブルリスト
ASSEMBLER END	翻訳終了メッセージ
Ready CLEAR 300,&H6FFF Feady	更に続けて翻訳を行なわないので，F-BASICの使用する作業領域の上限，文字列領域の大きさを標準に戻す.

図 8.2　ディスクの場合の翻訳手順例

```
Ready
AUTO 1000
1000 ' NAM BLKTFR
1010 ' OPT SYM
1020 ' ORG $5000
1030 'BLOCK LEAX SOURCE,PCR
1040 ' LEAY DEST,PC
1050 'BLOCK1 LDA 0,X+
1060 ' STA 0,Y+
1070 ' CMPA #4
1080 ' BNE BLOCK1
        :
2050 'SOURCE FCC/12345/
2060 ' FCB 4
2070 'DEST RMB 6
2080 ' END
2090

Ready
```

F−BASICのAUTUコマンドを用いて，行番号が1000からのソースプログラムをメモリ上に作成する.

```
LIST

1000 ' NAM BLKTFR
1010 ' OPT SYM
1020 ' ORG $5000
1030 'BLOCK LEAX SOURCE,PCR
1040 ' LEAY DEST,PC
1050 'BLOCK1 LDA 0,X+
1060 ' STA 0,Y+
1070 ' CMPA #4
1080 ' BNE BLOCK1
        :
2050 'SOURCE FCC/12345/
2060 ' FCB 4
2070 'DEST RMB 6
2080 ' END
```

作成したソースプログラムをF−BASICのLISTコマンドを用いて，リスト出力させ各ステートメントの目視チェックを行なう.

```
Ready
EDIT 1040

1040 ' LEAY DEST,PC (修正前)
```

リストチェックの結果，行番号1040のオペランドに誤りが見つかったためF−BASICのEDITコマンドを用いて修正を行なう.

PC ———→ PCR
(誤)　修正　(正)

```
1040 ' LEAY DEST,PCR (修正後)
SAVE "BLKTFR",A
```

修正されたソースプログラムをSAVEコマンドを用いて，ASCII形式でファイルに格納する(ディスク).

図8.3　ソースプログラムファイルの作成例

8.4 アセンブラプログラムのコピー方法

(1) 富士通提供のアセンブラプログラムが格納されているディスク，すなわちコピーするディスクをディスクドライブユニット #0 にセットし，コピーされるディスクをディスクドライブユニット #1 にセットします.

(2) CLEAR 300,&H17FF
LOAD "ASM09"
LOADM "ASM09EB"
これらのコマンドにより，アセンブラプログラムが，ディスクドライブユニット #0 のディスクからメモリへロードされます.

(3) SAVE "1：ASM09"
SAVEM "1：ASM09EB",&H1800,&H3DFF,&H1800
これらのコマンドにより，"ASM09"というBASICプログラムファイル及び"ASM09EB"という機械語プログラムファイルが，ディスクドライブユニット #1 にセットされているディスク上に格納されて，コピーは終了します.

8.5 アセンブルリストの種類

アセンブルリストは，ソースプログラムとアセンブラによって生成された追加情報からなります．ここでは，アセンブルリストとその見方について説明します.

アセンブルリストは，制御データ(「8.2　翻訳の手順」参照)の出力装置の指定で出力するかどうかの選択ができます．出力するときは更に，CRT ディスプレイとプリンタのどちらに出力するか選択が可能です.

8.5.1 ソースプログラム及び目的プログラムリスト

ソースプログラム及び目的プログラムリストは，ソースプログラムの各々の文と，それに対応した目的コードを同時に出力したものです.

目的コードは，マイクロプロセッサでプログラムを実行する場合のそのままの形をしており，実行時の論理的な誤りを見付け出すのに有効です.

また，ソースプログラムの翻訳中にエラーが発見されたとき，アセンブラは，自動的にエラー行の前にエラー番号を印刷し，エラー件数を最後に印刷します．エラー番号については「8.6.1　エラーメッセージ」を参照してください．

次にソースプログラム及び目的プログラムリストの標準的な形式を表8.1に示します．

表8.1　ソースプログラム及び目的プログラムリストの形式

桁位置	説明
1～5	ソースプログラムの文の行番号（10進）
6～7	空白
8～11	ロケーションカウンタの値
12	空白
13～14	機械命令コードの上位バイト
15～16	機械命令コードの下位バイト（もしあれば）
17	空白
	分岐命令，インデックス命令でないとき
18～19	オペランドの上位バイト
20～21	オペランドの下位バイト（もしあれば）
22～62	空白
	インデックス命令のとき
18～19	ポストバイト
20	空白
21～22	オペランドの上位バイト
23～24	オペランドの下位バイト（もしあれば）
25～26	空白
	分岐命令の場合
18～19	分岐命令オフセットの上位バイト
20～21	分岐命令オフセットの下位バイト（もしあれば）
22	空白
23～26	分岐先の絶対番地
	BSZ/EQU/ORG命令等の場合
18～19	式の値の上位バイト
20～21	式の値の下位バイト（もしあれば）
22～26	空白
27	空白
28～33	ラベル欄
34	空白
35～40	命令欄
41	空白
42～48	オペランド欄（長くなる場合には注釈欄も使用する）
49	空白
50～79	注釈欄

表8.1に示したようにソースプログラムは，翻訳されて印刷されるときは，アセンブラにより編集されて出力されます．従って，注釈欄に記述された注釈は，最大29文字まで印刷されます(ただし，OPT命令のLLENオプションで一行に印刷する文字数を79桁以上指定できる場合にはこの限りではない)．この文字数を越えて注釈を記述しても翻訳エラーとはなりませんが，リストには印刷されません．

しかし注釈行は，ソースプログラムに記述された桁位置のまま28桁目から印刷されます．

8.5.2 記号テーブルリスト

記号テーブルリストは，ソースプログラムのOPT命令で，SYMオプションの指定を行なうことにより出力されます．この指定が行なわれるとアセンブラは，記号テーブルにあるすべての記号を番地とともに印刷します．

8.6 エラー及び警告メッセージ

アセンブラは，翻訳時にエラー及び警告を検出すると，アセンブルリストにエラーメッセージ及び警告メッセージを出力します．

8.6.1 エラーメッセージ

エラーメッセージは，通常そのエラー行の直前に次のような形式でエラー番号(173～247)とリンク(直前に起ったエラーの文の行番号)が印刷されます．

```
****ERROR   eee--nnnn
 (注) eee   ：エラー番号
      nnnnn：エラーのリンク
```

通常，翻訳エラーが発生しても翻訳作業は続行され，目的プログラムも出力されます．しかし，そのままでは実行できませんので，F-BASICのMONコマンドを用いてメモリに格納されているプログラムを直接メモリ上で修正するか，ソースプログラムをF-BASICのエディット機能を用いて修正して再翻訳しなければなりません．

次にエラー番号と対応するエラーメッセージ一覧を表8.2に示します．

表8.2　エラーメッセージ一覧（つづく）

エラー番号	説　　　明
173	直接形式指定子の使用エラー 直接形式指定子“＜”が拡張間接番地指定形式で指定された.
174	自動インクリメント／デクリメントの形式エラー 自動インクリメント（＋1），或は自動デクリメント（－1）が間接形式で指定された．または，プラス或はマイナス記号を三つ以上検出した.
175	インデックス番地指定のエラー インデックス番地指定のアキュムレータオフセット形式で，X，Y，S，Uレジスタ以外のレジスタを指定した.
176	PSH/PUL命令(1)エラー PSHS，PULS，PSHU，PULU命令に続く直接式で，REG命令以外で定義された記号を検出したか，“!+”以外の演算子を検出した，或は“#”の後に記号の指定がなにもない.
177	PSH/PUL命令(2)エラー PSHS命令又はPULS命令で，レジスタ並び中にSレジスタを検出した. PSHU命令又はPULU命令で，レジスタ並び中にUレジスタを検出した. REG命令で，SレジスタとUレジスタの両方を同時に指定している.
178	レジスタ指定(1)のエラー レジスタ並び中に未定義レジスタを検出した. TFR命令，EXG命令でちょうど2個のレジスタを指定していない（1個以下或は3個以上のレジスタを指定）. PSH/PUL命令で，レジスタを全く指定していない.
179	レジスタ指定(2)のエラー EXG命令で指定した2つのレジスタの大きさが等しくない. TFR命令で8ビットレジスタから16ビットレジスタの転送を指定している.
200	ソースプログラム異常終了 ソースプログラムの翻訳中，END命令を処理する前にEOF(End of File)を検出した.
202	ラベル又は命令コードエラー ラベル又は命令コードが，英字あるいはピリオドで始まっていない.

表8.2　エラーメッセージ一覧（つづき）

エラー番号	説　　　　明
203	文の記述エラー 行番号につづく引用符の記述がしていない．
205	ラベルエラー 文のラベル欄が，一つ以上の空白で終ってない．
206	ORG 命令エラー ORG 命令が，オーバーラップしている．
207	未定義命令コードエラー 定義されていない簡略命令コードを使用している．
208	分岐命令（ショート分岐命令）のエラー 分岐命令のオフセット値が，分岐可能な範囲を越えている． 許される範囲は， （＊＋2）－128≦M≦（＊＋2）＋127 （注）＊：分岐命令の先頭番地 M：分岐命令の分岐先の番地
209	番地指定のエラー この命令では許されていない番地指定である．
210	1バイトオーバフロー 文で指定されている命令のオペランドの数値，記号式の値が，－128より小さいか又は256より大きい．
211	未定義記号エラー 記号がラベル欄に定義されていない．
214	FCB 命令エラー FCB 命令が，文法的に正しくない．
215	FDB 命令エラー FDB 命令が，文法的に正しくない．
216	オペランドエラー 命令のオペランドにエラーがある．

表8.2　エラーメッセージ一覧（つづき）

エラー番号	説　　　　明
217	OPT 命令エラー OPT 命令がエラーか，未定義オプションが指定されている．
220	フェーズエラー パス1とパス2で機械語命令のラベルに割り当てられたロケーションカウンタの値が不一致である．
221	記号テーブルオーバフロー 記号テーブルがオーバフローした．これは致命的なエラーであり，翻訳は中断されBASICに制御が移る．
222	記号の文法エラー 記号として，プログラマが使用できない特殊記号（A，B，CC，D，DP，PC，PCR，S，U，X，Y）を使っている．
223	ラベルエラー 必ずラベルを付けなければならないか，ラベルを付けてはならない命令なのにそのようになっていない．
226	カッコの対応エラー 左右の括弧の数が一致していない．
227	数値表現エラー 数値の自己規定項の表現にエラーがあるか，あるいは数値の評価にオーバフローが発生した．
233	記号名エラー 使用されている記号名が6文字を越えている．
234	記号名2重定義エラー 既に定義されている記号を定義しようとしている．又は，2重定義されている記号名を参照しようとした．
235	記憶域エラー OPT命令で，MEMオプション指定がなされているとき，目的コードをアセンブラプログラムの上に書き込みを行なおうとした．あるいは，指定された記憶域には，メモリが実装されていなかった．

表 8.2　エラーメッセージ一覧（つづき）

エラー番号	説　　明
236	ロケーションカウンタオーバフロー ロケーションカウンタがオーバフローした．又は，ロケーションカウンタが昇順になっていない．
241	記号使用エラー 未定義記号あるいは，まだ定義されていない記号など許されない記号を使用している．
244	ページ，リスト長指定エラー OPT 命令のPAGE，リスト長の指定が，許されている範囲を越えている．（10≦ページサイズ≦255，50≦ライン長≦136）
247	オペランド終了エラー オペランドの次は通常空白，あるいはCRコードでなければならないのにそのようになっていない．

通常翻訳エラー番号は，エラー行の前に印刷されますが，例外的にエラー行の後に印刷されることもあります．それは次の行が来るまで，エラーかどうか判定できないようなときです．図 8.4 に例を示します．

```
00030   0000        2A      MSG    FCC    1*** BATA ERROR ***/
00040   0012        0001    ERR    RMB    1
00050   0013        29             FCC    />/
00060   0014        30             FCC    $04
****ERROR     216--00000
00070   0016        04             FCB    $04
00080               0017    START  EQU    *
00090   0017        001A           LDX    PACK
```

図 8.4　例外的エラー行の印刷の例

図 8.4 は，行番号60のFCC命令でオペランド欄の'$'を区切り記号とみなし展開しようとしましたが，対になる区切り記号が見付からずエラーとなったものです．

8.6.2 警告メッセージ

ＯＰＴ命令でＷオプションを指定することにより，アセンブラは次のような形で警告が起ったことをプログラマに知らせます.

＊＊＊WARNING www－－nnnnn

（注）www　：警告番号

nnnnn：直前に起った警告の文の行番号

次に，警告番号と対応する警告メッセージ一覧を表8.3に示します.

表8.3　警告メッセージ一覧

警告番号	説明
1	ロング分岐命令にしなくてもよい. ロング分岐命令が－126から＋129の範囲内の番地へ分岐するのに用いられた.
2	拡張番地指定を用いるべきである. 直接番地指定が指定子“＜”を用いて強制されたが，SETDP命令により割り当てられた直接ページ擬似レジスタは拡張番地指定を用いるべきであることを示している.
3	レジスタを重複して指定している. 同じレジスタ名を，レジスタ並び中に二回以上指定した.（例えば，レジスタAあるいはBと一緒にレジスタDを指定したなど）
4	SETDP命令の式が誤っている. SETDP命令の式の最上位バイトの値が0でない. しかし，直接ページ擬似レジスタにはとにかく式の最下位バイトの値が割り当てられる.
5	転送エラー TFR命令でオペランド欄に16ビットレジスタから8ビットレジスタへの転送を指定した. このとき，16ビットレジスタの上位バイトは無視され，下位バイトが8ビットレジスタへ転送される.

8.7 翻訳時のメモリ配置

アセンブラが，ソースプログラムの翻訳を行なうときの，メモリ配置を図 8.5 に示します.

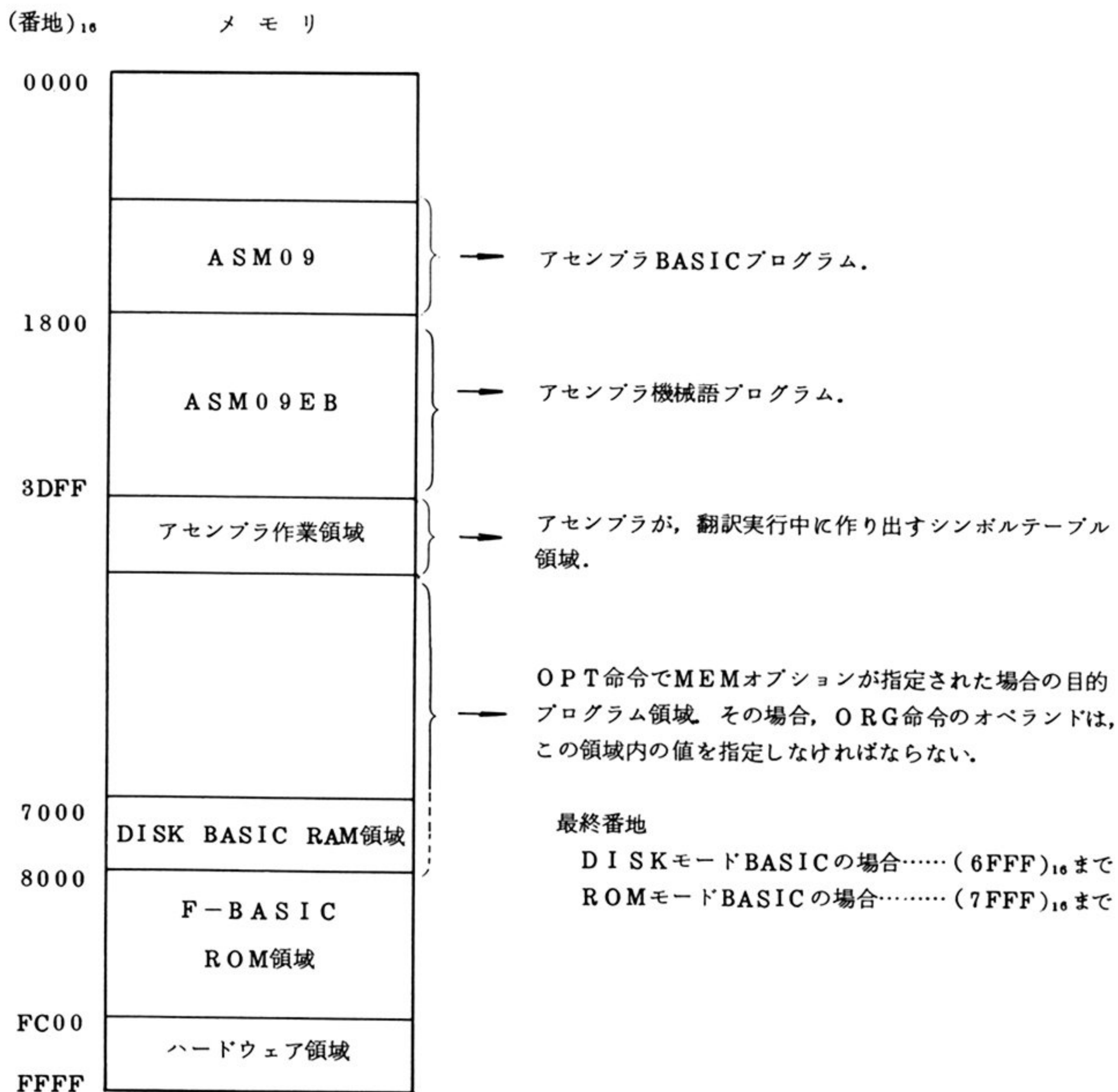



図8.5 翻訳時のメモリ配置図

8.8 翻訳処理能力

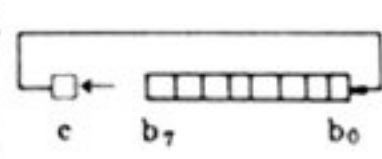
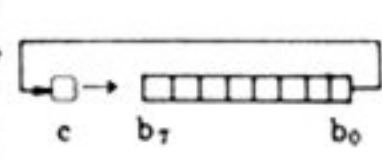
アセンブラは，記号テーブルのオーバフローが発生しない限り，ソースプログラムの翻訳時ステップ数の制限はありません．

ただし，ロケーションカウンタがオーバフローしたときは，この限りではありません．

記号テーブルは，ソースプログラムで定義された記号を格納し，1個の記号に対して10バイト必要とします．

付録1　機械命令一覧表

命令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長（バイト）	機能	E	F	H	I	N	Z	V	C
Add B-accumulator to X	ABX	空き	IN	3A	3	1	B＋X→X (Unsigned)	•	•	•	•	•	•	•	•
Add memory to accumulator with carry	ADCA	#e_2	IM	89	2	2	A＋M＋C→A	•	•	①	•	②	④	⑤	⑩
		e_1	DI	99	4	2									
		e_5, R	IX	A9	ⓒ4＋	ⓒ2＋									
		e_3	EX	B9	5	3									
	ADCB	#e_2	IM	C9	2	2	B＋M＋C→B								
		e_1	DI	D9	4	2									
		e_5, R	IX	E9	4＋	2＋									
		e_3	EX	F9	5	3									
Add memory to accumulator	ADDA	#e_2	IM	8B	2	2	A＋M→A	•	•	①	•	②	④	⑤	⑩
		e_1	DI	9B	4	2									
		e_5, R	IX	AB	4＋	2＋									
		e_3	EX	BB	5	3									
	ADDB	#e_2	IM	CB	2	2	B＋M→B								
		e_1	DI	DB	4	2									
		e_5, R	IX	EB	4＋	2＋									
		e_3	EX	FB	5	3									
	ADDD	#e_4	IM	C3	4	3	D＋M：M＋1→D	•	•	•	•	③	④	⑥	⑪
		e_1	DI	D3	6	2									
		e_5, R	IX	E3	6＋	2＋									
		e_3	EX	F3	7	3									
AND Memory with accumulator	ANDA	#e_2	IM	84	2	2	A⋂M→A	•	•	•	•	②	④	R	•
		e_1	DI	94	4	2									
		e_5, R	IX	A4	4＋	2＋									
		e_3	EX	B4	5	3									
	ANDB	#e_2	IM	C4	2	2	B⋂M→B								
		e_1	DI	D4	4	2									
		e_5, R	IX	E4	4＋	2									
		e_3	EX	F4	5	3									
AND condition code register	ANDCC	#e_2	IM	1C	3	2	CC⋂IMM→CC	⑳	⑳	⑳	⑳	⑳	⑳	⑳	⑳
Arithmetic shift of accumulator or memory left	ASL	A	A	48	2	1	A	•	•	⑲	•	②	④	⑦	⑫
		B	B	58	2	1	B								
		e_1	DI	08	6	2	} □←□□□□□□□□←0 (c　b_7 ← b_0)								
		e_5, R	IX	68	6＋	2＋	M								
		e_3	EX	78	7	3									

命令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長(バイト)	機能	E	F	H	I	N	Z	V	C
Arithmetic shift of accumulator or memory right	ASR	A	A	47	2	1	A	•	•	⑲	•	②	④	•	⑬
		B	B	57	2	1	B								
		e_1	DI	07	6	2	M (b7 → b0 → c)								
		e_5, R	IX	67	6+	2+									
		e_3	EX	77	7	3									
Branch if carry clear	BCC	e_0	RE	24	3	2	Branch C=0	•	•	•	•	•	•	•	•
	LBCC	e_4	RE	1024	ⓑ5(6)	4	Long Branch C=0								
Branch if carry set	BCS	e_0	RE	25	3	2	Branch C=1	•	•	•	•	•	•	•	•
	LBCS	e_4	RE	1025	ⓑ5(6)	4	Long Branch C=1								
Branch if equal	BEQ	e_0	RE	27	3	2	Branch Z=1	•	•	•	•	•	•	•	•
	LBEQ	e_4	RE	1027	ⓑ5(6)	4	Long Branch Z=1								
Branch if greater than or equal (signed)	BGE	e_0	RE	2C	3	2	Branch≧Zero N⊕V=0	•	•	•	•	•	•	•	•
	LBGE	e_4	RE	102C	ⓑ5(6)	4	Long Branch≧Zero N⊕V=0								
Branch if greater (signed)	BGT	e_0	RE	2E	3	2	Branch>Zero Z∪(N⊕V)=0	•	•	•	•	•	•	•	•
	LBGT	e_4	RE	102E	ⓑ5(6)	4	Long Branch>Zero Z∪(N⊕V)=0								
Branch if higher (unsigned)	BHI	e_0	RE	22	3	2	Branch Higher C∪Z=0	•	•	•	•	•	•	•	•
	LBHI	e_4	RE	1022	ⓑ5(6)	4	Long Branch higher C∪Z=0								
Branch if higher or same (unsigned)	BHS	e_0	RE	24	3	2	Branch C=0	•	•	•	•	•	•	•	•
	LBHS	e_4	RE	1024	ⓑ5(6)	4	Long Branch C=0								
Bit test memory with accumulator	BITA	#e_2	IM	85	2	2	Bit Test A (M∩A)	•	•	•	•	②	④	R	•
		e_1	DI	95	4	2									
		e_5, R	IX	A5	4+	2+									
		e_3	EX	B5	5	3									
	BITB	#e_2	IM	C5	2	2	Bit Test B (M∩B)								
		e_1	DI	D5	4	2									
		e_5, R	IX	E5	4+	2+									
		e_3	EX	F5	5	3									
Branch if less than or equal (signed)	BLE	e_0	RE	2F	3	2	Branch Z∪(N⊕V)=1	•	•	•	•	•	•	•	•
	LBLE	e_4	RE	102F	ⓑ5(6)	4	Long Branch Z∪(N⊕V)=1								
Branch if lower (unsigned)	BLO	e_0	RE	25	3	2	Branch Lower C=1	•	•	•	•	•	•	•	•
	LBLO	e_4	RE	1025	ⓑ5(6)	4	Long Branch C=1								
Branch if lower or same (unsigned)	BLS	e_0	RE	23	3	2	Branch C∪Z=1	•	•	•	•	•	•	•	•
	LBLS	e_4	RE	1023	ⓑ5(6)	4	Long Branch C∪Z=1								
Branch if less than (signed)	BLT	e_0	RE	2D	3	2	Branch<Zero N⊕V=1	•	•	•	•	•	•	•	•
	LBLT	e_4	RE	102D	ⓑ5(6)	4	Long Branch<Zero N⊕V=1								
Branch if minus	BMI	e_0	RE	2B	3	2	Branch Minus N=1	•	•	•	•	•	•	•	•
	LBMI	e_4	RE	102B	ⓑ5(6)	4	Long Branch Minus N=1								
Branch if not equal	BNE	e_0	RE	26	3	2	Branch Z=0	•	•	•	•	•	•	•	•
	LBNE	e_4	RE	1026	ⓑ5(6)	4	Long Branch Z=0								
Branch if plus	BPL	e_0	RE	2A	3	2	Branch Plus N=0	•	•	•	•	•	•	•	•
	LBPL	e_4	RE	102A	ⓑ5(6)	4	Long Branch Plus N=0								
Branch always	BRA	e_0	RE	20	3	2	Branch Always	•	•	•	•	•	•	•	•
	LBRA	e_4	RE	16	5	3	Long Branch Always								

命令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長 (バイト)	機能	E	F	H	I	N	Z	V	C
Branch never	BRN	e_0	RE	21	3	2	Branch Never	•	•	•	•	•	•	•	•
	LBRN	e_4	RE	1021	5	4	Long Branch Never								
Branch to subroutine	BSR	e_0	RE	8D	7	2	Branch to Subroutine	•	•	•	•	•	•	•	•
	LBSR	e_4	RE	17	9	3	Long Branch to Subroutine								
Branch if overflow clear	BVC	e_0	RE	28	3	2	Branch V=0	•	•	•	•	•	•	•	•
	LBVC	e_4	RE	1028	ⓑ5(6)	4	Long Branch V=0								
Branch if overflow set	BVS	e_0	RE	29	3	2	Branch V=1	•	•	•	•	•	•	•	•
	LBVS	e_4	RE	1029	ⓑ5(6)	4	Long Branc V=1								
Clear accumulator or memory location	CLR	A	A	4F	2	1	0→A	•	•	•	•	R	S	R	R
		B	B	5F	2	1	0→B								
		e_1	DI	0F	6	2	0→M								
		e_5, R	IX	6F	6+	2+									
		e_3	EX	7F	7	3									
Compare memory from accumulator	CMPA	✚e_2	IM	81	2	2	A−M	•	•	⑲	•	②	④	⑤	⑭
		e_1	DI	91	4	2									
		e_5, R	IX	A1	4+	2+									
		e_3	EX	B1	5	3									
	CMPB	✚e_2	IM	C1	2	2	B−M								
		e_1	DI	D1	4	2									
		e_5, R	IX	E1	4+	2+									
		e_3	EX	F1	5	3									
	CMPD	✚e_4	IM	1083	5	4	D−M:M+1	•	•	•	•	③	④	⑥	⑮
		e_1	DI	1093	7	3									
		e_5, R	IX	10A3	7+	3+									
		e_3	EX	10B3	8	4									
Compare memory from index register	CMPX	✚e_4	IM	8C	4	3	X−M:M+1	•	•	•	•	③	④	⑥	⑮
		e_1	DI	9C	6	2									
		e_5, R	IX	AC	6+	2+									
		e_3	EX	BC	7	3									
	CMPY	✚e_4	IM	108C	5	4	Y−M:M+1								
		e_1	DI	109C	7	3									
		e_5, R	IX	10AC	7+	3+									
		e_3	EX	10BC	8	4									
Compare memory from stack pointer	CMPS	✚e_4	IM	118C	5	4	S−M:M+1	•	•	•	•	③	④	⑥	⑮
		e_1	DI	119C	7	3									
		e_5, R	IX	11AC	7+	3+									
		e_3	EX	11BC	8	4									
	CMPU	✚e_4	IM	1183	5	4	U−M:M+1								
		e_1	DI	1193	7	3									
		e_5, R	IX	11A3	7+	3+									
		e_3	EX	11B3	8	4									

命令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長(バイト)	機能	E	F	H	I	N	Z	V	C
Complement accumlator or memory location	COM	A	A	43	2	1	$\overline{A}$→A	•	•	•	•	②	④	R	S
		B	B	53	2	1	$\overline{B}$→B								
		e_1	DI	03	6	2	$\overline{M}$→M								
		e_5, R	IX	63	6+	2+									
		e_3	EX	73	7	3									
AND condition code register, then wait for interrupt	CWAI	♦e_2	IM	3C	20	2	CC⋂IMM →CC, Wait for interrupt	S	⑳	⑳	⑳	⑳	⑳	⑳	⑳
Decimal adjust, A-accumulator	DAA	空き	IN	19	21	1	BCD形式の2進加算結果をBCD形式に変換	•	•	•	•	②	④	⑲	⑰
Decrement accumlator or memory location	DEC	A	A	4A	2	1	A−1→A	•	•	•	•	②	④	⑧	•
		B	B	5A	2	1	B−1→B								
		e_1	DI	0A	6	2	M−1→M								
		e_5, R	IX	6A	6+	2+									
		e_3	EX	7A	7	3									
Exclusive or memory with accumulator	EORA	♦e_2	IM	88	2	2	A⊕M→A	•	•	•	•	②	④	R	•
		e_1	DI	98	4	2									
		e_5, R	IX	A8	4+	2+									
		e_3	EX	B8	5	3									
	EORB	♦e_2	IM	C8	2	2	B⊕M→B								
		e_1	DI	D8	4	2									
		e_5, R	IX	E8	4+	2+									
		e_3	EX	F8	5	3									
Exchange R1 with R2	EXG	R_1, R_2	RG	1E	8	2	R_1↔R_2 ⓓ	㉑	㉑	㉑	㉑	㉑	㉑	㉑	㉑
Increment accumulator to memory location	INC	A	A	4C	2	1	A+1→A	•	•	•	•	②	④	⑨	•
		B	B	5C	2	1	B+1→B								
		e_1	DI	0C	6	2	M+1→M								
		e_5, R	IX	6C	6+	2+									
		e_3	EX	7C	7	3									
Jump	JMP	e_1	DI	0E	3	2	EA→PC (注) EA:実行アドレス	•	•	•	•	•	•	•	•
		e_5, R	IX	6E	3+	2+									
		e_3	EX	7E	4	3									
Jump to subroutine	JSR	e_1	DI	9D	7	2	SP←SP−1, (SP)←PC_L SP←SP−1, (SP)←PC_H PC←EA (注) EA:実行アドレス	•	•	•	•	•	•	•	•
		e_5, R	IX	AD	7+	2+									
		e_3	EX	BD	8	3									
Load accumulator from memory	LDA	♦e_2	IM	86	2	2	M→A	•	•	•	•	②	④	R	•
		e_1	DI	96	4	2									
		e_5, R	IX	A6	4+	2+									
		e_3	EX	B6	5	3									
	LDB	♦e_2	IM	C6	2	2	M→B								
		e_1	DI	D6	4	2									
		e_5, R	IX	E6	4+	2+									
		e_3	EX	F6	5	3									

命令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長(バイト)	機能	E	F	H	I	N	Z	V	C
	LDD	#e_4	IM	CC	3	3	M:M+1→D	•	•	•	•	③	④	R	•
		e_1	DI	DC	5	2									
		e_5, R	IX	EC	5+	2+									
		e_3	EX	FC	6	3									
Load index register from memory	LDX	#e_4	IM	8E	3	3	M:M+1→X	•	•	•	•	③	④	R	•
		e_2	DI	9E	5	2									
		e_5, R	IX	AE	5+	2+									
		e_3	EX	BE	6	3									
	LDY	#e_4	IM	108E	4	4	M:M+1→Y								
		e_1	DI	109E	6	3									
		e_5, R	IX	10AE	6+	3+									
		e_3	EX	10BE	7	4									
Load stack pointer from memory	LDS	#e_4	IM	10CE	4	4	M:M+1→S	•	•	•	•	③	④	R	•
		e_1	DI	10DE	6	3									
		e_5, R	IX	10EE	6+	3+									
		e_3	EX	10FE	7	4									
	LDU	#e_4	IM	CE	3	3	M:M+1→U								
		e_1	DI	DE	5	2									
		e_5, R	IX	EE	5+	2+									
		e_3	EX	FE	6	3									
Load effective address into index register	LEAX	e_5, R	IX	30	4+	2+	EA→X (注)EA:実行アドレス	•	•	•	•	•	④	•	•
	LEAY	e_5, R	IX	31	4+	2+	EA→Y (注)EA:実行アドレス								
Load effective address into stack register	LEAS	e_5, R	IX	32	4+	2+	EA→S (注)EA:実行アドレス	•	•	•	•	•	•	•	•
	LEAU	e_5, R	IX	33	4+	2+	EA→U (注)EA:実行アドレス								
Logical shift left accumulator or memory location	LSL	A	A	48	2	1	A, B, M: □← □□□□□□□□ ←0 (c ← b_7 ← b_0)	•	•	⑲	•	②	④	⑦	⑫
		B	B	58	2	1									
		e_1	DI	08	6	2									
		e_5, R	IX	68	6+	2									
		e_3	EX	78	7	3									
Logical shift right accumulator or memory location	LSR	A	A	44	2	1	A, B, M: 0→ □□□□□□□□ →□ (b_7 → b_0 c)	•	•	•	•	R	④	•	⑬
		B	B	54	2	1									
		e_1	DI	04	6	2									
		e_5, R	IX	64	6+	2									
		e_3	EX	74	7	3									
Unsigned multiply	MUL	空き	IN	3D	11	1	A＊B→D	•	•	•	•	•	④	•	⑱
Negate accumulator or memory	NEG	A	A	40	2	1	$\overline{A}$+1→A	•	•	⑲	•	②	④	⑧	⑭
		B	B	50	2	1	$\overline{B}$+1→B								
		e_1	DI	00	6	2	$\overline{M}$+1→M								
		e_5, R	IX	60	6+	2+									
		e_3	EX	70	7	3									
No operation	NOP	空き	IN	12	2	1	PC+1→PC	•	•	•	•	•	•	•	•
OR memory with accumulator	ORA	#e_2	IM	8A	2	2	A∪M→A	•	•	•	•	②	④	R	•
		e_1	DI	9A	4	2									
		e_5, R	IX	AA	4+	2+									
		e_3	EX	BA	5	3									

命　令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長（バイト）	機　能	E	F	H	I	N	Z	V	C
OR memory with accumulator	ORB	#e₂	IM	CA	2	2	B⋃M→B	•	•	•	•	②	④	R	•
		e₁	DI	DA	4	2									
		e₅ , R	IX	EA	4＋	2＋									
		e₃	EX	FA	5	3									
OR condition code register	ORCC	#e₂	IM	1A	3	2	CC⋃IMM→CC	⑳	⑳	⑳	⑳	⑳	⑳	⑳	⑳
Push any register(s) onto hardware stack	PSHS	R₁〔,…, Rn〕	RG	34	ⓐ5＋	2	ビット7が1ならば； SP←SP−1,(SP)←PC_L SP←SP−1,(SP)←PC_H ビット6が1ならば； SP←SP−1,(SP)←US_L SP←SP−1,(SP)←US_H ビット5が1ならば； SP←SP−1,(SP)←Y_L SP←SP−1,(SP)←Y_H ビット4が1ならば； SP←SP−1,(SP)←X_L SP←SP−1,(SP)←X_H ビット3が1ならば； SP←SP−1,(SP)←DP ビット2が1ならば； SP←SP−1,(SP)←B ビット1が1ならば； SP←SP−1,(SP)←A ビット0が1ならば； SP←SP−1,(SP)←CC	•	•	•	•	•	•	•	•
Push any register(s) onto user stack	PSHU	R₁〔,…, Rn〕	RG	36	ⓐ5＋	2	ビット7が1ならば； US←US−1,(US)←PC_L US←US−1,(US)←PC_H ビット6が1ならば； US←US−1,(US)←SP_L US←US−1,(US)←SP_H ビット5が1ならば； US←US−1,(US)←Y_L US←US−1,(US)←Y_H ビット4が1ならば； US←US−1,(US)←X_L US←US−1,(US)←X_H ビット3が1ならば； US←US−1,(US)←DP ビット2が1ならば； US←US−1,(US)←B ビット1が1ならば； US←US−1,(US)←A ビット0が1ならば； US←US−1,(US)←CC	•	•	•	•	•	•	•	•

命令	簡略コード	オペランド		命令コード	実行サイクル	命令長(バイト)	機能	コンディションコード							
		記述形式	番地指定					E	F	H	I	N	Z	V	C
Pull any register(s) from hardware stack	PULS	R₁〔,…, Rn〕	RG	35	ⓐ5＋	2	ビット0が1ならば； (SP)→CC ,SP+1→SP ビット1が1ならば； (SP)→A ,SP+1→SP ビット2が1ならば； (SP)→B ,SP+1→SP ビット3が1ならば； (SP)→DP ,SP+1→SP ビット4が1ならば； (SP)→X_H ,SP+1→SP (SP)→X_L ,SP+1→SP ビット5が1ならば； (SP)→Y_H ,SP+1→SP (SP)→Y_L ,SP+1→SP ビット6が1ならば； (SP)→US_H ,SP+1→SP (SP)→US_L ,SP+1→SP ビット7が1ならば； (SP)→PC_H ,SP+1→SP (SP)→PC_L ,SP+1→SP	㉑	㉑	㉑	㉑	㉑	㉑	㉑	㉑
Pull any register(s) from user stack	PULU	R₁〔,…, Rn〕	RG	37	ⓐ5＋	2	ビット0が1ならば； (US)→CC ,US+1→US ビット1が1ならば； (US)→A ,US+1→US ビット2が1ならば； (US)→B ,US+1→US ビット3が1ならば； (US)→DP ,US+1→US ビット4が1ならば； (US)→X_H ,US+1→US (US)→X_L ,US+1→US ビット5が1ならば； (US)→Y_H ,US+1→US (US)→Y_L ,US+1→US ビット6が1ならば； (US)→SP_H ,US+1→US (US)→SP_L ,US+1→US ビット7が1ならば； (US)→PC_H ,US+1→US (US)→PC_L ,US+1→US	㉑	㉑	㉑	㉑	㉑	㉑	㉑	㉑

命令	簡略	オペランド		命令	実行	命令長	機能	コンディションコード							
	コード	記述形式	番地指定	コード	サイクル	(バイト)		E	F	H	I	N	Z	V	C
Rotate accumulator or memory left	ROL	A	A	49	2	1	A B M c b$_7$ b$_0$	•	•	•	•	②	④	⑦	⑫
		B	B	59	2	1									
		e_1	DI	09	6	2									
		e_5 , R	IX	69	6+	2+									
		e_3	EX	79	7	3									
Rotate accumulator or memory right	ROR	A	A	46	2	1	A B M c b$_7$ b$_0$	•	•	•	•	②	④	•	⑬
		B	B	56	2	1									
		e_1	DI	06	6	2									
		e_5 , R	IX	66	6+	2+									
		e_3	EX	76	7	3									
Return from interrupt	RTI	空き	IN	3B	6/15	1	(SP)→CC, SP+1→SP CCのEフラグが1ならば (SP)→A ,SP+1→SP (SP)→B ,SP+1→SP (SP)→DP ,SP+1→SP (SP)→X_H ,SP+1→SP (SP)→X_L ,SP+1→SP (SP)→Y_H ,SP+1→SP (SP)→Y_L ,SP+1→SP (SP)→US_H ,SP+1→SP (SP)→US_L ,SP+1→SP (SP)→PC_H ,SP+1→SP (SP)→PC_L ,SP+1→SP CCのEフラグが0ならば (SP)→PC_H ,SP+1→SP (SP)→PC_L ,SP+1→SP	⑳	⑳	⑳	⑳	⑳	⑳	⑳	⑳
Return from subroutine	RTS	空き	IN	39	5	1	(SP)→PC_H ,SP+1→SP (SP)→PC_L ,SP+1→SP	•	•	•	•	•	•	•	•
Subtract memory from accumulator with borrow	SBCA	✚e_2	IM	82	2	2	A−M−C→A	•	•	⑲	•	②	④	⑤	⑭
		e_1	DI	92	4	2									
		e_5 , R	IX	A2	4+	2+									
		e_3	EX	B2	5	3									
	SBCB	✚e_2	IM	C2	2	2	B−M−C→B								
		e_1	DI	D2	4	2									
		e_5 , R	IX	E2	4+	2+									
		e_3	EX	F2	5	3									
Sign Extend B-accumulator into A-accumlator	SEX	空き	IN	ID	2	1	AccBのb_7=1なら$(FF)_{16}$→A AccBのb_7=0なら 0 →A	•	•	•	•	②	④	•	•
Store accumulator to memory	STA	e_1	DI	97	4	2	A→M	•	•	•	•	②	④	R	•
		e_5 , R	IX	A7	4+	2+									
		e_3	EX	B7	5	3									

命令	簡略コード	オペランド 記述形式	オペランド 番地指定	命令コード	実行サイクル	命令長(バイト)	機能	E	F	H	I	N	Z	V	C
Store accumulator to memory	STB	e_1	DI	D7	4	2	B→M								
		e_5 ,R	IX	E7	4+	2+		•	•	•	•	②	④	R	•
		e_3	EX	F7	5	3									
	STD	e_1	DI	DD	5	2	D→M：M+1								
		e_5 ,R	IX	ED	5+	2+		•	•	•	•	③	④	R	•
		e_3	EX	FD	6	3									
Store index register to memory	STX	e_1	DI	9F	5	2	X→M：M+1								
		e_5 ,R	IX	AF	5+	2+									
		e_3	EX	BF	6	3		•	•	•	•	③	④	R	•
	STY	e_1	DI	109F	6	3	Y→M：M+1								
		e_5 ,R	IX	10AF	6+	3+									
		e_3	EX	10BF	7	4									
Store stack pointer to memory	STS	e_1	DI	10DF	6	3	S→M：M+1								
		e_5 ,R	IX	10EF	6+	3+									
		e_3	EX	10FF	7	4		•	•	•	•	③	④	R	•
	STU	e_1	DI	DF	5	2	U→M：M+1								
		e_5 ,R	IX	EF	5+	2+									
		e_3	EX	FF	6	3									
Subtract memory from accumulator	SUBA	#e_2	IM	80	2	2	A−M→A								
		e_1	DI	90	4	2									
		e_5 ,R	IX	A0	4+	2+									
		e_3	EX	B0	5	3		•	•	⑲	•	②	④	⑤	⑩
	SUBB	#e_2	IM	C0	2	2	B−M→B								
		e_1	DI	D0	4	2									
		e_5 ,R	IX	E0	4+	2+									
		e_3	EX	F0	5	3									
	SUBD	#e_4	IM	83	4	3	D−M：M+1→D								
		e_1	DI	93	6	2		•	•	•	•	③	④	⑥	⑯
		e_5 ,R	IX	A3	6+	2+									
		e_3	EX	B3	7	3									
Software interrupt (absolute indirect)	SWI	空き	IN	3F	19	1	Eフラグをセット SP←SP−1,(SP)←PC_L SP←SP−1,(SP)←PC_H SP←SP−1,(SP)←US_L SP←SP−1,(SP)←US_H SP←SP−1,(SP)←Y_L SP←SP−1,(SP)←Y_H SP←SP−1,(SP)←X_L SP←SP−1,(SP)←X_H SP←SP−1,(SP)←DP SP←SP−1,(SP)←B SP←SP−1,(SP)←A SP←SP−1,(SP)←CC IフラグとFフラグをセット PC←($FFFA):($FFFB)	S	S	•	S	•	•	•	•

命　　令	簡略コード	オペランド 記述形式	番地指定	命令コード	実行サイクル	命令長(バイト)	機　　能	E	F	H	I	N	Z	V	C
Software interrupt (absolute indirect)	SWI2	空き	IN	103F	20	2	Eフラグをセット SP←SP−1,(SP)←PC_L SP←SP−1,(SP)←PC_H SP←SP−1,(SP)←US_L SP←SP−1,(SP)←US_H SP←SP−1,(SP)←Y_L SP←SP−1,(SP)←Y_H SP←SP−1,(SP)←X_L SP←SP−1,(SP)←X_H SP←SP−1,(SP)←DP SP←SP−1,(SP)←B SP←SP−1,(SP)←A SP←SP−1,(SP)←CC PC←($FFF4):($FFF5)	S	•	•	•	•	•	•	•
	SWI3	空き	IN	113F	20	2	Eフラグをセット SP←SP−1,(SP)←PC_L SP←SP−1,(SP)←PC_H SP←SP−1,(SP)←US_L SP←SP−1,(SP)←US_H SP←SP−1,(SP)←Y_L SP←SP−1,(SP)←Y_H SP←SP−1,(SP)←X_L SP←SP−1,(SP)←X_H SP←SP−1,(SP)←DP SP←SP−1,(SP)←B SP←SP−1,(SP)←A SP←SP−1,(SP)←CC PC←($FFF2):($FFF3)	S	•	•	•	•	•	•	•
Synchronize with interrupt line	SYNC	空き	IN	13	≧2	1	命令の実行を停止する	•	•	•	•	•	•	•	•
Transfer R_1 to R_2	TFR	R_1, R_2	RG	1F	6	2	R_1→R_2 ⓓ	㉑	㉑	㉑	㉑	㉑	㉑	㉑	㉑
Test	TST	A	A	4D	2	1	A−0	•	•	•	•	②	④	R	•
		B	B	5D	2	1	B−0								
		e_1	DI	0D	6	2	M−0								
		e_5, R	IX	6D	6+	2+									
		e_3	EX	7D	7	3									

記号の説明

(1) オペランドの記述形式

A, B, # ……… 記号そのものを表す.

e_0 ……… 値が－126から＋129の範囲の式.

e_1 ……… 値が0から255の範囲の式.

e_2 ……… 値が－128から255の範囲の式.

e_3 ……… 値が256以上の式.

e_4 ……… 一般の式を表す.

e_5 ……… 一般の式，又はA，B，Dレジスタのうちの一つを表す.

R ……… X，Y，S，U，PCRレジスタのうち一つを表す.

R_1, R_2, R_n …… A，B，CC，D，DP，PC，S，U，X，Yレジスタのうちの一つを表す.

(2) 番地指定形式

IN ……… インヘレント番地指定

A ……… アキュムレータ番地指定（アキュムレータA）

B ……… アキュムレータ番地指定（アキュムレータB）

RG ……… レジスタ番地指定

IM ……… イミーディエイト番地指定

IX ……… インデックス番地指定

RE ……… 相対番地指定

DI ……… 直接番地指定

EX ……… 拡張番地指定

(3) 演　算　子

←, →…… 矢印の方向にデータを転送する.

∩ ……… 論理積（AND）

∪ ……… 論理和（OR）

⊕ ……… 排他的論理和（Exclusive OR）

￣ ……… 論理否定

（　）……… 括弧内の内容を表す.

IMM …… イミーディエイト値

M ……… 命令オペランドで示されたメモリ領域の番地

(4) コンディション・コード

E ……… エンタイアフラグ

F ……… FIRQマスクフラグ

H ……… ハーフキャリーフラグ

I ……… IRQマスクフラグ

N ……… 負の表示フラグ

Z ……… ゼロの表示フラグ

V ……… 2の補数のオーバ・フローの表示フラグ

C ……… キャリー又はボローの表示フラグ

(5) コンディションコードの状態

① ……… 演算により，ビット3からキャリーが生じるとセットされる.

② ……… 結果により，ビット7から1ならばセットされる.

③ ……… 結果により，ビット15から1ならばセットされる.

④ ……… 結果により，すべてのビットが0ならばセットされる.

⑤ ……… 演算により，8ビットの2の補数のオーバフローが生じるとセットされる.

⑥ ……… 演算により，16ビットの2の補数のオーバフローが生じるとセットされる.

⑦ ……… 演算の前のオペランドの($b_7 \oplus b_6$)の結果が入る.

⑧ ……… 演算前のオペランドば2進の10000000ならばセットされる.

⑨ ……… 演算前のオペランドが2進の01111111ならばセットされる.

⑩ ……… 演算により，ビット7からキャリーが生じるとセットされる.

⑪ ……… 上位バイトの演算でビット7からキャリーが生じるとセットされる.

⑫ ……… 演算前のオペランドのビット7が入る.

⑬ ……… 演算前のオペランドのビット0が入る.

⑭ ……… 演算により，ビット7からキャリーが生じないとセットされる.

⑮ ……… 演算により，ビット15からキャリーが生じないとセットされる.

⑯ ……… 上位バイトの演算で，ビット7からキャリーが生じないとセット

される．

⑰ ………演算よりビット7からキャリーが生じるか，演算前にCフラグがセットされていたらセットされる．

⑱ ………演算の結果，AccBのビット7が1ならばセットされる．

⑲ ………この命令が実行された後のフラグの値は意味をもたない（不定）．

⑳ ………コンディションコードは，命令実行の結果に従ってセットされる．

㉑ ………オペランドにCCが指定された場合には命令実行の結果に従ってセットされます．しかし，その他の場合には変化しません．

• ………変化しない．

R ………リセット

S ………セット

(6) その他

ⓐ ………PSH/PUL命令の所用マシンサイクル数は，『5＋（転送するレジスタのバイト数）』になる．

ⓑ ………5(6)は，ブランチしないときのサイクル数が5で，ブランチしたときのサイクル数は6になることを示している．

ⓒ ………インデックス番地指定のサイクル数，命令長の“＋”は，アドレスモードによりそれぞれ増加されることを示している．増加分については，「付録2　ポスバイトの形式」参照．

ⓓ ………EXG命令とTFR命令のR_1とR_2は，各々8ビットレジスタどうしか，16ビットレジスタどうしである．

8ビットレジスタ：A，B，CC，DP

16ビットレジスタ：X，Y，S，U，D，PC

付録2　ポストバイトの形式

(1)　インデックスアドレシングのポストバイト

アドレッシング・モード			インダイレクトでない場合				インダイレクトの場合			
			アセンブラ形式	ポストバイト	+~	+#	アセンブラ形式	ポストバイト	+~	+#
インデックス	式オフセット	オフセットなし	,R	1RR00100	0	0	〔,R〕	1RR10100	3	0
		5ビット・オフセット	n,R	0RRnnnnn	1	0	−	−	−	−
		8ビット・オフセット	n,R	1RR01000	1	1	〔n,R〕	1RR11000	4	1
		16ビット・オフセット	n,R	1RR01001	4	2	〔n,R〕	1RR11001	7	2
	アキュムレータオフセット	Aレジスタ・オフセット	A,R	1RR00110	1	0	〔A,R〕	1RR10110	4	0
		Bレジスタ・オフセット	B,R	1RR00101	1	0	〔B,R〕	1RR10101	4	0
		Dレジスタ・オフセット	D,R	1RR01011	4	0	〔D,R〕	1RR11011	7	0
	自動インクリメント/デクリメント	インクリメント(+1)	,R+	1RR00000	2	0	−	−	−	−
		インクリメント(+2)	,R++	1RR00001	3	0	〔,R−−〕	1RR10001	6	0
		デクリメント(−1)	,−R	1RR00010	2	0	−	−	−	−
		デクリメント(−2)	,−−R	1RR00011	3	0	〔,−−R〕	1RR10011	6	0
プログラムカウンタ相対		8ビット・オフセット	n,PCR	1××01100	1	1	〔,PCR〕	1××11100	4	1
		16ビット・オフセット	n,PCR	1××01101	5	2	〔n,PCR〕	1××11101	8	2
拡張間接		16ビット・アドレス	−	−	−	−	〔n〕	10011111	5	2

〔記号の説明〕

R　：X，Y，S，Uレジスタのうちの一つ.

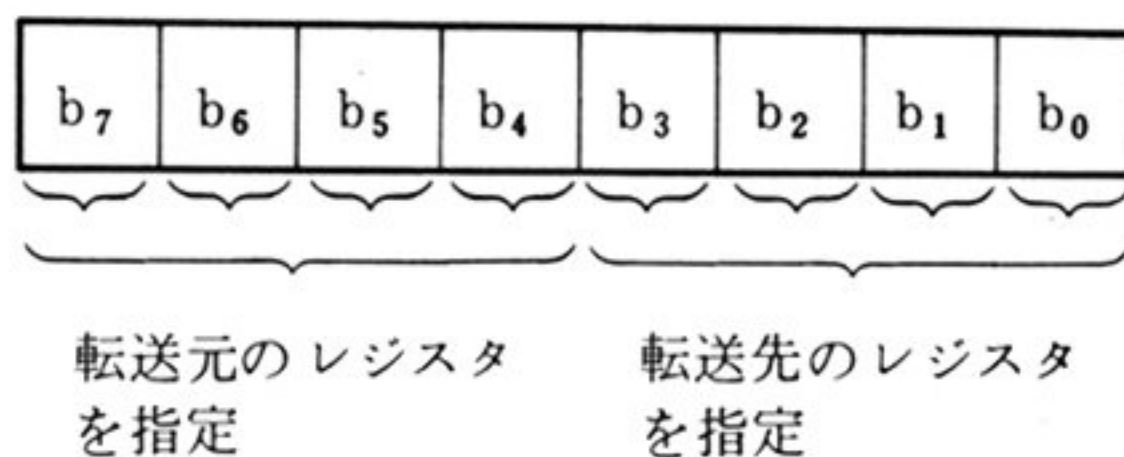
RR：指定されたレジスタを表す.

Xレジスタの場合　00
Yレジスタの場合　01
Uレジスタの場合　10
Sレジスタの場合　11

X　：任　意

+~　：追加されるマシンサイクル数

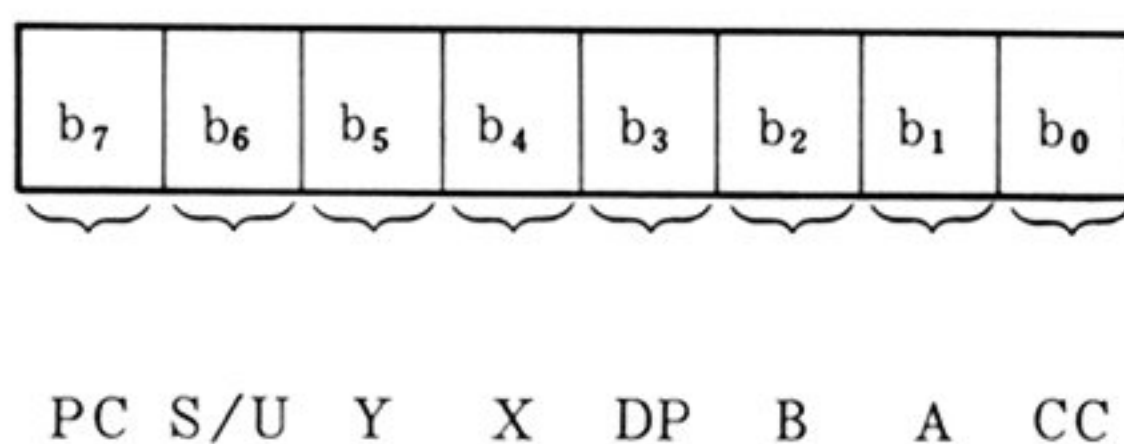
+#　：追加されるバイト数

(2) TFR／EXG命令のポストバイト

各々のレジスタのビットパターンは次のとおり．

D＝0000　　PC＝0101
X＝0001　　A ＝1000
Y＝0010　　B ＝1001
U＝0011　　CC＝1010
S＝0100　　DP＝1011

(3) PSH／PUL命令のポストバイト

レジスタによってビットの位置が決まっていて，最上位ビット b_7 はPCレジスタを，ビット b_6 はPSHU及びPULU命令のときはSレジスタを，PSHS 及び PULS命令のときはUレジスタを，ビット b_5 はYレジスタを，ビット b_4 はXレジスタを，ビット b_3 はDPレジスタを，ビット b_2 はBレジスタを，b_1 はAレジスタを，最下位ビット b_0 はCCレジスタをそれぞれ表し，オペラント欄で指定されるとそれぞれのビットは1がたつ．

PC　S/U　Y　X　DP　B　A　CC

付録3　アセンブラ命令一覧表

アセンブラ命令には，以下に説明する記号が使用されています．

［　］……括弧内を省略することもできる．　・・・……直前の項目の反復を示す．

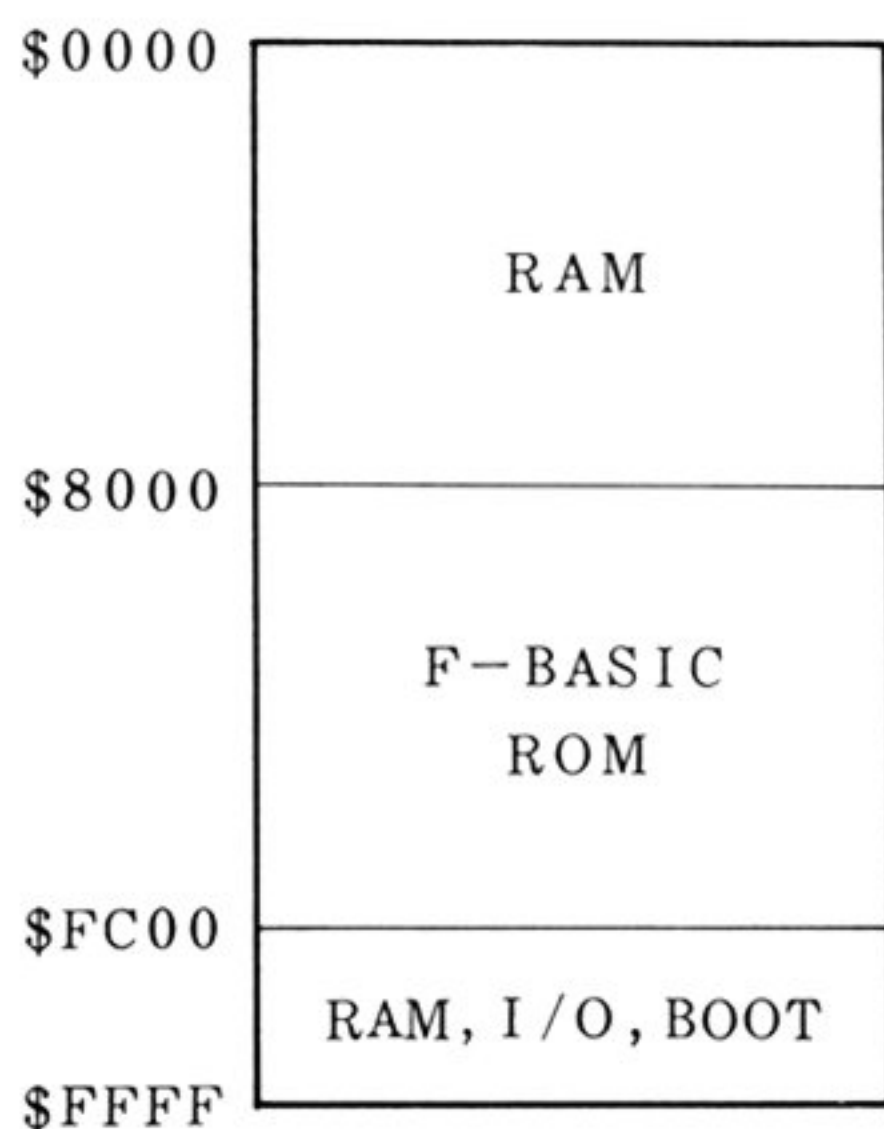
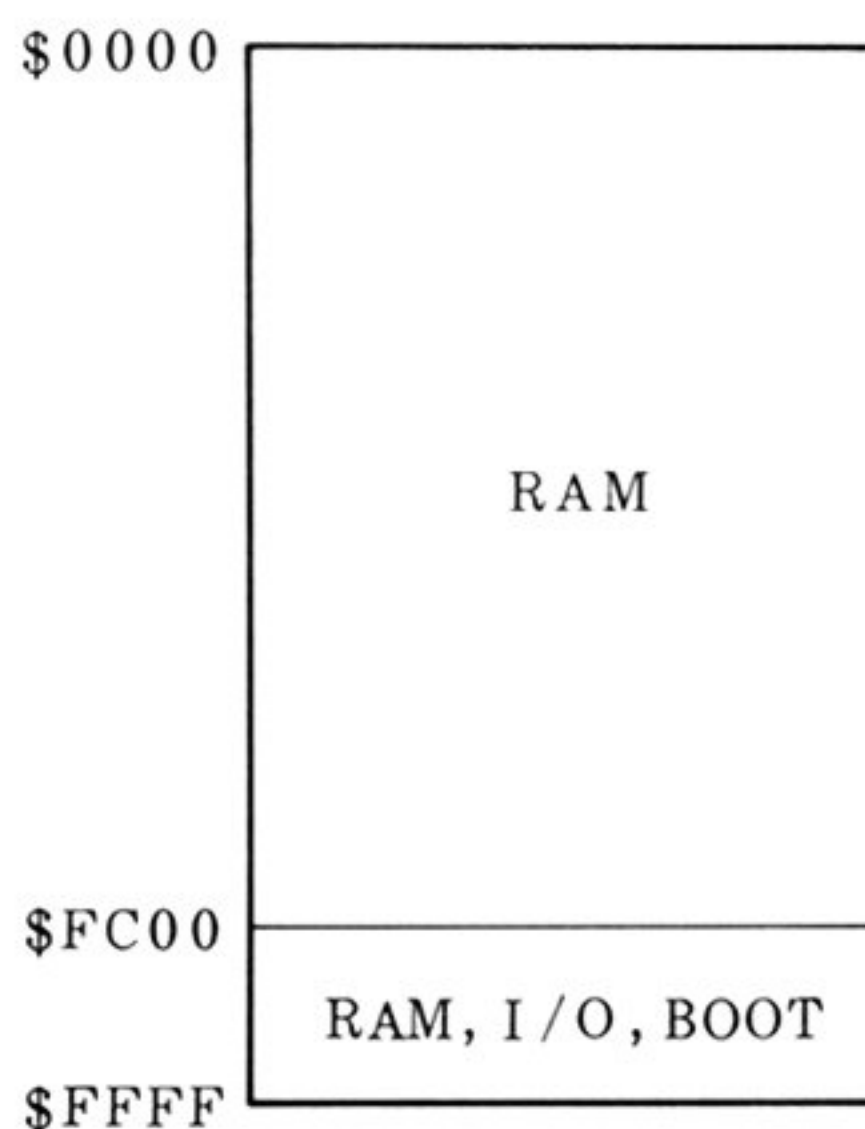
｛　｝……括弧内の一つの項目を選択する．　……省略時解釈を示す．

また，空欄は，その欄に指定する項がないことを示す．

ラベル欄の記述形式	簡易命令コード	オペランド欄の記述形式	命令の機能
［記号］	BSZ	式	初期値が$(00)_{16}$の領域の確保
	END	［式］	ソースプログラム単位の終了
記号	EQU	式	記号の定義
［記号］	FCB	｛｛式, | 空白,｝［式, | ,］［・・・］［式］ | 式｝	1バイト定数の定義
［記号］	FCC	｛文字数，文字列 | 区切り記号　文字列　区切り記号｝	文字データの定数
［記号］	FDB	｛｛式, | 空白,｝［式, | ,］［・・・］［式］ | 式｝	2バイト定数の定義
	NAM	プログラム名	ソースプログラム単位の開始
	OPT	［｛O［BJ］ | NOO［BJ］｝］［,｛M［EM］ | NOM［EM］｝］ ［,LLE［N］＝n］［,｛L［IST］ | NOL［IST］｝］ ［,P［AGE］＝n］［,｛G［EN］ | NOG［EN］｝］ ［｛,S［YM］ | ,NOS［YM］｝］［,｛W | NOW｝］	出力形式の制御
	ORG	式	ロケーションカウンタ値変更
	PAG［E］		改ページ指定
記号	REG	レジスタ並び	レジスタ並びを指定
［記号］	RMB	式	領域の確保
	SETDP	式	直接ページ擬似レジスタ変更
	SPC	式	空白行の作成
	TTL	文字列	見出し行の作成

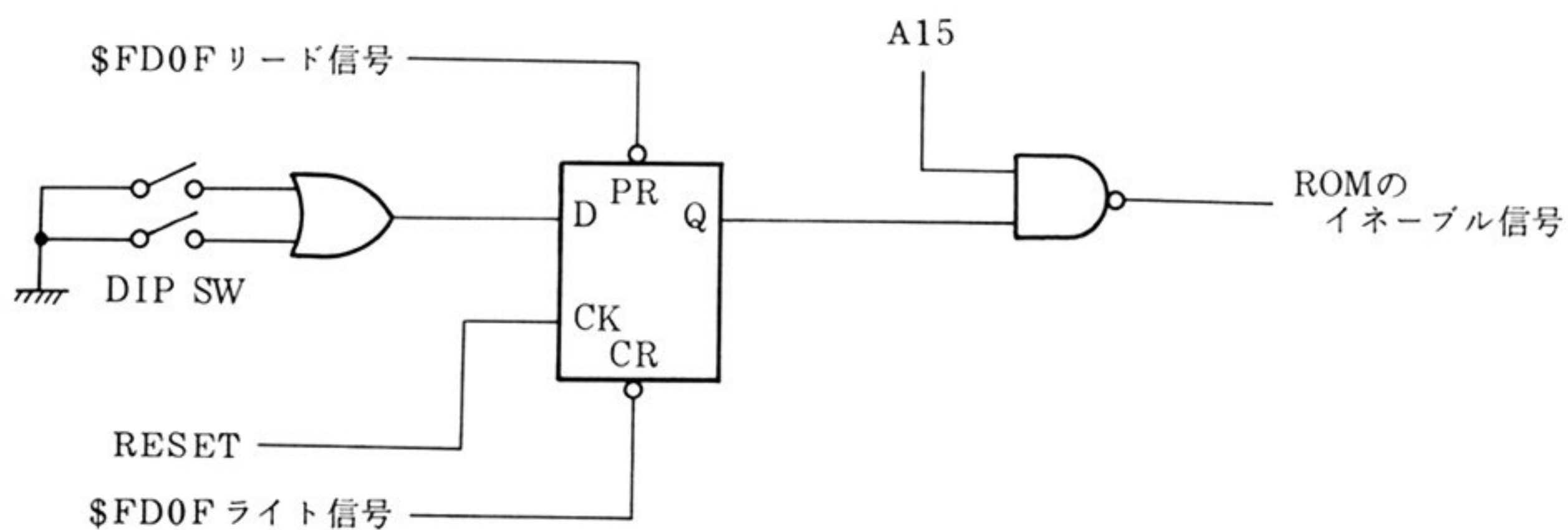
付録4　裏RAM機能

付録4.1　裏RAMについて

FM−7のF−BASIC使用時におけるメモリ構成は，通常，アドレス＄8000～＄FBFFまではF−BASIC ROMが接続されていますが，プログラムによってF−BASIC ROMを切り離してオールRAMモードにてプログラムを実行させることもできます．ただし，オールRAMモードに切換えた場合には，F−BASICの機能を一切使用することはできないため，すべて機械語にて処理しなくてはなりません．

F−BASIC ROMモード時　　　　オールRAMモード時

F−BASIC ROMモードとオールRAMモードとの切換えは＄FD0F番地をリード/ライトすることによって行ないます．＄FD0F番地をリードすることによってF−BASIC ROMモードに，＄FD0F番地をライトすることによってオール RAMモードに設定されますが，その切換えはかならず＄0000～＄7FFFF番地内の機械語プログラムによって行なわなければなりません．＄8000番地以降にてF−BASIC ROMモードに戻したり，BASICのPOKE文などで切換えたりしますと，プログラムは必ず暴走します．

付録4.2　裏RAM制御回路

裏RAMの制御は，BASIC ROMのイネーブル信号を制御することによって行ないます．以下に，その切替え部分の回路を示します．

ROMのイネーブル信号を反転した信号がROMイネーブル信号線にも接続されており，ROMをイネーブルすると同時に裏RAMはディスエーブルされ，ROMをディスエーブルすると同時に裏RAMはイネーブルされます．

付録4.3　裏RAMにデータやプログラムを転送する方法

F−BASICにて裏RAMの部分にデータやプログラム（機械語）を転送するためには，一度，F−BASICのフリーエリア（＄0000〜＄7FFF）にLOADM文などで，データやプログラム（機械語）を読み込んでおいてから機械語プログラムを用いて，オールRAMモードにして，裏RAMにデータを転送します．

裏RAMにデータを転送するための機械語プログラム（RAMEB）の例を以下に示します．

```
PAGE   001  (830513,160300)  EX-1

00100                                   NAM    EX-1
00110                                   TTL    ** DATA <---> ｳﾗRAM **
00120   6000                            ORG    $6000
00130               6000        START   EQU    *
00140   6000 20     08    600A          BRA    ENTRY
00150   6002        0002        FROM    RMB    2        * TRANCE-START ADDRESS
00160   6004        0002        TO      RMB    2        * RECEIVE-START ADDRESS
00170   6006        0002        LEN     RMB    2        * DATA LENGTH
00180   6008        0000        LAST    FDB    0        * TRANCE-END ADDRESS
00190                           *                       *
00200                           *** MAIN PROGRAM ***
00210                           *                       *
00220               600A        ENTRY   EQU    *
00230   600A EC     8C F5               LDD    FROM,PCR
00240   600D 1F     01                  TFR    D,X      * TRANCE-START ADDRESS --> IX
00250   600F E3     8C F4               ADDD   LEN,PCR
00260   6012 FD     6008                STD    LAST     * TRANCE-END ADDRESS --> LAST
00270   6015 10AE   8C EB               LDY    TO,PCR   * RECEIVE-START ADDRESS --> IY
00280   6019 B7     FD0F                STA    $FD0F    * ｳﾗRAM ｲﾈｰﾌﾞﾙ
00290   601C A6     80          LOOP    LDA    ,X+      * DATA TRANCE
00300   601E A7     A0                  STA    ,Y+
00310   6020 AC     8C E5               CMPX   LAST,PCR
00320   6023 26     F7    601C          BNE    LOOP     * DATA-END CHECK
00330   6025 B6     FD0F                LDA    $FD0F    * BASIC-ROM ｲﾈｰﾌﾞﾙ
00340   6028 39                         RTS             * RETURN TO BASIC
00350               6000                END    START
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=6000
PROGRAM END   ADDR=6028
PROGRAM ENTRY ADDR=6000
```

この機械語プログラム（RAMEB）を読み込んだ後，F－BASICのPOKE文で転送データが格納されている先頭アドレス（FROM）と受信データを格納する先頭アドレス（TO）と転送データのバイト数（LEN）の値をセットします．このプログラムは，F－BASICのフリーエリアから裏RAMへデータを転送する場合だけでなく，裏RAMからF－BASICのフリーエリアへデータを転送する場合も使用できます．ただし，FROM，TO，LENの値のチェックは一切行なっていないので，値の設定は十分注意して行なって下さい．

F－BASICによる機械語プログラムRAMEBの使用例を以下に示します．

```
100 ' DATA <---> ウラRAM
110 ' EX-1 ····· SAMPLE PROGRAM
120 CLEAR 300,&H4FFF
130 LOADM "RAMEB"
140 EXADD=&H6000
150 ' DATA SET
160 FOR I=0 TO 255
170 POKE &H5000+I,I
180 NEXT I
190 ' TRANCE TO ウラRAM
200 POKE EXADD+2,&H50    ' TRANCE-START HIGH-ADDRESS
210 POKE EXADD+3,&H00    ' TRANCE-START LOW-ADDRESS
220 POKE EXADD+4,&H80    ' RECEIVE-START HIGH-ADDRESS
230 POKE EXADD+5,&H00    ' RECEIVE-START LOW-ADDRESS
240 POKE EXADD+6,&H01    ' DATA LENGTH HIGH-ADDRESS
250 POKE EXADD+7,&H00    ' DATA LENGTH LOW-ADDRESS
260 EXEC EXADD
270 ' RECEIVE FROM ウラRAM
280 POKE EXADD+2,&H80    ' TRANCE-START HIGH-ADDRESS
290 POKE EXADD+3,&H00    ' TRANCE-START LOW-ADDRESS
300 POKE EXADD+4,&H51    ' RECEIVE-START HIGH-ADDRESS
310 POKE EXADD+5,&H00    ' RECEIVE-START LOW-ADDRESS
320 POKE EXADD+6,&H01    ' DATA LENGTH HIGH-ADDRESS
330 POKE EXADD+7,&H00    ' DATA LENGTH LOW-ADDRESS
340 EXEC EXADD
350 END
```

このBASICプログラムは，160～180行のFOR NEXTループで$5000番地から256バイトに0～255のデータを設定してから，そのデータを200～260行のプログラムで裏RAMにデータを転送します．さらに，280～340行のプログラムで裏RAMに転送したデータを$5100番地から256バイトに転送します．

このプログラムを実行する前にメモリの内容をF－BASICのMONコマンドで調べると，以下のようになっています（電源投入後すぐにプログラムを入れた場合）．

```
MON

*D5000

5000 00 00 00 00 00 00 00 00
5008 00 00 00 00 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*D5100

5100 00 00 00 00 00 00 00 00
5108 00 00 00 00 00 00 00 00
5110 00 00 00 00 00 00 00 00
5118 00 00 00 00 00 00 00 00
5120 00 00 00 00 00 00 00 00
5128 00 00 00 00 00 00 00 00
5130 00 00 00 00 00 00 00 00
5138 00 00 00 00 00 00 00 00
*
```

プログラム実行後のメモリの内容は，次のようになります．

```
RUN

Ready
MON

*D5000

5000 00 01 02 03 04 05 06 07
5008 08 09 0A 0B 0C 0D 0E 0F
5010 10 11 12 13 14 15 16 17
5018 18 19 1A 1B 1C 1D 1E 1F
5020 20 21 22 23 24 25 26 27
5028 28 29 2A 2B 2C 2D 2E 2F
5030 30 31 32 33 34 35 36 37
5038 38 39 3A 3B 3C 3D 3E 3F
*D5100
```

160～180行のFOR NEXTループでセットしたデータ

```
5100 00 01 02 03 04 05 06 07
5108 08 09 0A 0B 0C 0D 0E 0F
5110 10 11 12 13 14 15 16 17
5118 18 19 1A 1B 1C 1D 1E 1F
5120 20 21 22 23 24 25 26 27
5128 28 29 2A 2B 2C 2D 2E 2F
5130 30 31 32 33 34 35 36 37
5138 38 39 3A 3B 3C 3D 3E 3F
*
```

200～260行のプログラムによって裏RAMに転送したデータを，280～340のプログラムによって，裏RAMからフリーエリアに転送したデータ

〔注意〕 BASICのMONコマンドでは，$8000番地以降の裏RAMの部分を見ることはできません.

付録4.4 裏RAMのサブルーチンを使用する方法

F－BASICから裏RAM内のサブルーチンを呼び出すには，まず，F－BASICのフリーエリア内の機械語ルーチンにて裏RAMをイネーブル状態にしておいてから，裏RAM内のサブルーチンを呼び出します．また，F－BASICに戻る時にも，F－BASICのフリーエリア内の機械語ルーチンにてBASIC ROMをイネーブルにしてから，F－BASICに戻るという手順をとる必要があります．

以下に，F－BASICからの引数が整数型であれば1を加え，整数型でなければ何も処理をしないというサブルーチンの例を用いて解説します．

裏RAM内のサブルーチン（SUBEB）は次のようになります．

```
PAGE  001 (821201,002249) SUB-1

00100                                 NAM    SUB-1
00110                                 TTL    ** SUBROUTINE ON ｳﾗRAM **
00120  6100                           ORG    $6100
00130             6100         START  EQU    *
00140  6100 81    02                  CMPA   #$02      * INTEGER CHECK
00150  6102 26    07    610B          BNE    CONT
00160  6104 EC    02                  LDD    2,X
00170  6106 C3    0001                ADDD   #1        * ADD 1
00180  6109 ED    02                  STD    2,X
00190  610B 39                 CONT   RTS
00200             6100                END    START
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=6100
PROGRAM END   ADDR=610B
PROGRAM ENTRY ADDR=6100
```

SUBEBを読み出すためのプログラム（CALLEB）は次のようになります.

```
PAGE   001  (821201,002444)  EX-2

00100                                    NAM    EX-2
00110                                    TTL    ** CALL SUBROUTINE PROGRAM **
00120  6200                              ORG    $6200
00130               6200         START   EQU    *
00140  6200 20      02    6204           BRA    ENTRY
00150  6202         0002         SA      RMB    2          * SUB-START ADDRESS
00160  6204 F7      FD0F         ENTRY   STB    $FD0F      * ｳﾗRAM ｲﾈｰﾌﾞﾙ
00170  6207 AD      9C F8                JSR    [SA,PCR] * SUBROUTINE CALL
00180  620A F6      FD0F                 LDB    $FD0F      * BASIC-ROM ｲﾈｰﾌﾞﾙ
00190  620D 39                           RTS               * RETURN TO BASIC
00200               6200                 END    START
TOTAL  ERRORS 00000--00000
TOTAL  WARNINGS 00000--00000

PROGRAM BEGIN ADDR=6200
PROGRAM END   ADDR=620D
PROGRAM ENTRY ADDR=6200
```

このCALLEBプログラムは，F−BASICのフリーエリア内にて実行されなくてはなけません.

次に，SUBEB，CALLEBプログラムを動作させるためのF−BASIC プログラム例を示します.

```
100 ' ｳﾗRAM-SUBROUTINE CALL
110 ' EX-2 ····· SAMPLE PROGRAM
120 CLEAR 300,&H5FFF
130 LOADM "RAMEB"
140 LOADM "SUBEB"
150 EXADD=&H6000
160 ' RAMEB ---> ｳﾗRAM
170 POKE EXADD+2,&H61     ' SUBEB START HIGH-ADDRESS
180 POKE EXADD+3,&H00     ' SUBEB START LOW-ADDRESS
190 POKE EXADD+4,&H90     ' SUBEB TRANCE HIGH-ADDRESS
200 POKE EXADD+5,&H00     ' SUBEB TRANCE LOW-ADDRESS
210 POKE EXADD+6,&H00     ' SUBEB LENGTH HIGH-ADDRESS
220 POKE EXADD+7,&H0C     ' SUBEB LENGTH LOW-ADDRESS
230 EXEC &H6000
240 ' MAIN PROGRAM
250 LOADM "CALLEB"
260 EXADD2=&H6200
270 POKE EXADD2+2,&H90    ' SUBEB START HIGH-ADDRESS
280 POKE EXADD2+3,&H00    ' SUBEB START LOW-ADDRESS
290 DEFUSR1=EXADD2
300 DD%=8
310 DD%=USR1(DD%)
320 PRINT "DD% = ";DD%
330 END
```

付録4.5　BOISを使用する場合

裏RAM内の機械語ルーチンにてF−BASICのBIOSを使用する場合には，次の点に注意してプログラムして下さい.

① BIOSのRCB領域は，必ずF−BASICのフリーエリア内にて設定します.

② BIOSに渡すデータや渡されるデータは，必ずF−BASIC のフリーエリア内にて設定します.

③ BIOSを使用する前に，必ずF−BASICのフリーエリア内で，BASIC ROMをイネーブルにしてからBIOSをコールします．また，BIOSから戻った時は，再度裏RAMをイネーブルにしてから呼び出し元に帰るようにします.

BIOSを使用する時には，F−BASICのフリーエリア内に次のようなプログラムを入れておいて，BIOSを使用する際にこのプログラムを実行するようにすることをお勧めします.

```
PAGE  001  (821201,010429)  EX-3

00100                                  NAM    EX-3
00110                                  TTL    ** BIOS CALL PROGRAM **
00120  6300                            ORG    $6300
00130               6300       START   EQU    *
00140  6300 7D      FDOF               TST    $FDOF    * BASIC-ROM ｲﾈｰﾌﾞﾙ
00150  6303 AD      9F FBFA            JSR    [$FBFA]  * BIOS CALL
00160  6307 B7      FDOF               STA    $FDOF    * ｳﾗRAM ｲﾈｰﾌﾞﾙ
00170  630A 39                         RTS
00180               6300               END    START
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=6300
PROGRAM END   ADDR=630A
PROGRAM ENTRY ADDR=6300
```